

Introduction to Computer Graphics

Course Notes for SIGGRAPH '99

Course Organizer

Michael Bailey
University of California at San Diego, and
San Diego Supercomputer Center

Course Speakers

Andrew Glassner
Microsoft Research

Olin Lathrop
Cognivision, Inc.

Computer graphics is an exciting field of endeavor, but it is often difficult for a newcomer to get started. This course is that opportunity. The topics being presented will address many areas within computer graphics and treat each from the point of view of “why-do-I-care” and “how-to.” Those who take this course will emerge well-prepared to take on further study, including the taking of other SIGGRAPH courses. Attendees will also be ready to take on the vendor show and better appreciate the Electronic Theatre. We hope you enjoy reading and using these notes as much as we enjoyed preparing them.

If you have specific comments about how we can improve the course or the notes, please send them to me at: mjb@sdsc.edu

– Mike Bailey

Take them, use them, bring them to the masses.
Shake them, lose them, sing them to your classes.
Tiles of tides, piles of slides.
Piles of slides that no-one derides.
Slides of knowledge, slides of power – slides that last a half an hour.
Small slides. Blue slides. Old-hat and what's-new slides.
Take a slide and project it wide.
Project it far and make it tall, a slide's a slide that's seen by all.
SIGGRAPH slides go into holders, printed pages go into folders.
We teach. We teach in courses. We teach whatever the market enforces.
You want pixels? You want rays?
We'll lead you through the graphics maze.

– Andrew Glassner

SIGGRAPH '99

Introduction to Computer Graphics

About the Speakers

Michael J. Bailey

Mike Bailey is a researcher at the San Diego Supercomputer Center and a faculty member in Applied Mechanics / Engineering Sciences and Computer Science at the University of California at San Diego. Mike received his Ph.D. from Purdue University. He has also worked at Sandia National Laboratories, Purdue University, Megatek, SDSC, and UCSD. Mike's areas of interest include scientific visualization, computer aided design, and solid freeform fabrication. He has authored numerous papers on the use of computer graphics in engineering and science. Mike founded the interdisciplinary Design Visualization Lab at SDSC/UCSD, which includes the TeleManufacturing Facility which applies solid freeform fabrication methods to visualization problems. Mike has served on the SIGGRAPH Executive Committee and was SIGGRAPH conference co-chair in 1991. Mike has also served as SIGGRAPH Courses Chair in 1984, 1985, 1987, 1988, and 1994.

Andrew S. Glassner

Dr. Andrew Glassner is a Researcher at Microsoft Research, where he creates new computer graphics and new media. He has worked at the NYIT Computer Graphics Lab, Case Western Reserve University, the IBM TJ Watson Research Lab, the Delft University of Technology, Bell Communications Research, Xerox PARC, and Microsoft Research. He has published numerous technical papers on topics ranging from digital sound to new rendering techniques. His book *3D Computer Graphics: A Handbook for Artists and Designers* has taught a generation of artists. Glassner created and edited the *Graphics Gems* book series and the book *An Introduction to Ray Tracing*. His most recent text is *Principles of Digital Image Synthesis*, a two-volume treatise on rendering theory and practice published by Morgan-Kaufmann. Andrew served Siggraph '94 as Chair of the Papers Committee, and creator of the Sketches venue. He has also served as Founding Editor of the *Journal of Graphics Tools*, and Editor-in-Chief of *ACM Transactions on Graphics*. He directed the short animated film "Chicken Crossing" which premiered at the Siggraph '96 Electronic Theatre, and designed the highly participatory game "Dead Air" for The Microsoft Network. He has designed logos for electronics firms, publishers, and individuals. In his free time Andrew plays jazz piano, draws, and writes fiction. He holds a PhD in Computer Science from The University of North Carolina at Chapel Hill.

Olin Lathrop

Olin Lathrop works for Cognivision, Inc., where he does consulting and custom software development for computer graphics. Olin holds a Master of Engineering in Electrical Engineering from Rensselaer Polytechnic Institute. Olin has also worked at Hewlett-Packard, Raster Technologies, and Apollo Computer, where he specialized in graphics hardware design. Olin is the author of the introductory book *The Way Computer Graphics Works*.

SIGGRAPH '99

Introduction to Computer Graphics

Mike Bailey (M)
Andrew Glassner (A)
Olin Lathrop (L)

Course Schedule

| | |
|----------------------|--|
| 8:30 - 9:00 | WelcomeM Overview of the Course Some graphics to look at Overview of the Graphics Process |
| 9:00 – 10:00 | Modeling for Rendering and Animation.....A |
| 10:00 – 10:15 | Morning Break |
| 10:15 – 11:15 | Rendering.....A |
| 11:15 – 12:00 | Graphics display hardwareO |
| 12:00 – 1:30 | Lunch |
| 1:30 – 2:15 | Animation.....A |
| 2:15 – 3:00 | Geometry for computer graphics.....M |
| 3:00 – 3:15 | Afternoon Break |
| 3:15 – 3:30 | Input devicesM |
| 3:30 – 4:00 | Graphics on the World Wide Web.....M |
| 4:00 – 4:30 | Virtual realityO |
| 4:30 – 4:45 | Finding additional informationM |
| 4:45 – 5:00 | General Q&A.....All |

SIGGRAPH '99
Introduction to Computer Graphics

Course Note Table of Contents

- A. Introduction
- B. Overview of the Graphics Process
- C. An Introduction to Modeling
- D. 3D Object Modeling
- E. A Glossary for Modeling and Animation
- F. An Introduction to Rendering
- G. Graphics Display Hardware
- H. An Introduction to Animation
- I. Computer Animation Techniques
- J. Geometry for Computer Graphics
- K. Input Devices
- L. Graphics on the World Wide Web
- M. Virtual Reality
- N. Finding Additional Information
- O. Glossary of Computer Graphics Terms

Introduction to Computer Graphics

SIGGRAPH '99

Michael Bailey **University of California at San Diego and
San Diego Supercomputer Center**

Andrew Glassner **Microsoft Research**

Olin Lathrop **Cognivision, Inc.**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Mike Bailey

PhD from Purdue University

**Has worked at Sandia Labs, Purdue University,
Megatek, San Diego Supercomputer Center, and
the University of California at San Diego**

mjb@sdsc.edu



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Andrew Glassner

**PhD from the University of North Carolina -
Chapel Hill**

**Has worked at IBM, Bell Communications,
Delft University, NYIT, Xerox PARC, and
Microsoft Research**

glassner@microsoft.com



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Olin Lathrop

**Master of Engineering from Rensselaer
Polytechnic University**

**Has worked at Hewlett-Packard, Raster
Technologies, Apollo Computer, and
Cognivision**

olin@cognivis.com



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Course Goals

- **Provide a background for papers, panels, and other courses**
- **Help appreciate the Electronic Theater**
- **Get more from the vendor exhibits**
- **Give our take on where the future is**
- **Provide pointers for further study**



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Topics

- **Overview of the “Graphics Process” (Mike)**
- **Modeling (Andrew)**
- **Rendering (Andrew)**
- **Display Hardware (Olin)**



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

More Topics

- **Animation (Andrew)**
- **Geometry for Computer Graphics (Mike)**
- **Input Devices (Mike)**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

And, Even More Topics !

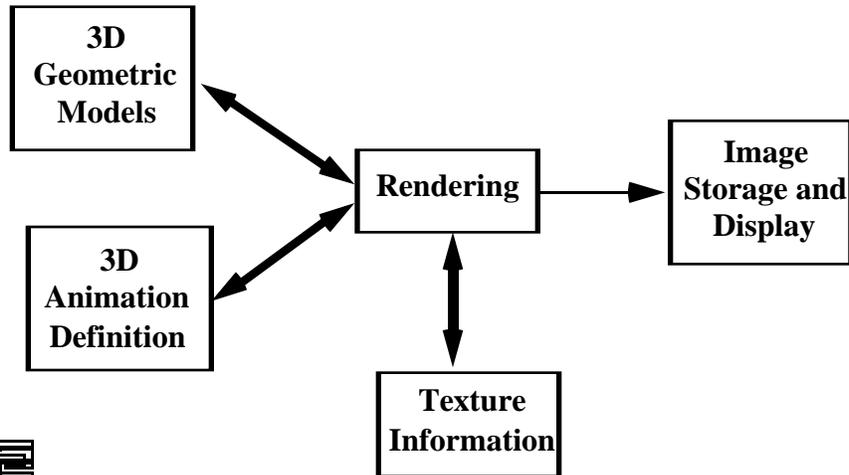
- **Graphics on the Web (Mike)**
- **Virtual Reality (Olin)**
- **Finding Additional Information (Mike)**



SAN DIEGO SUPERCOMPUTER CENTER

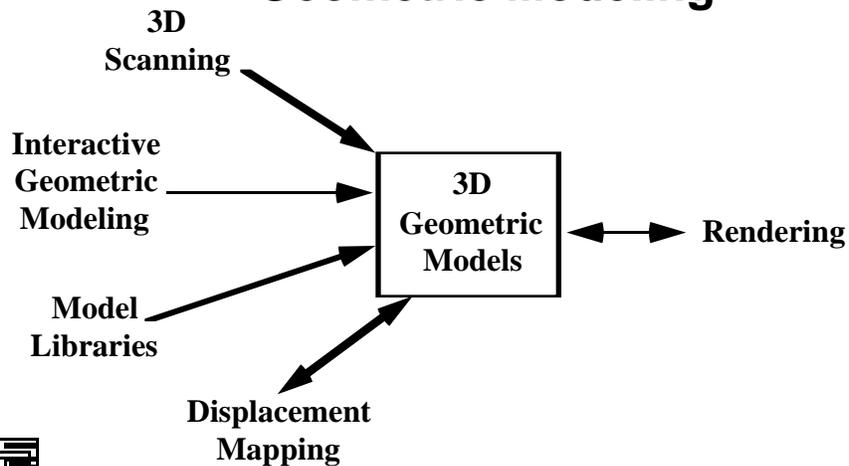
A National Center for Computational Science & Engineering

The Graphics Process: Summary



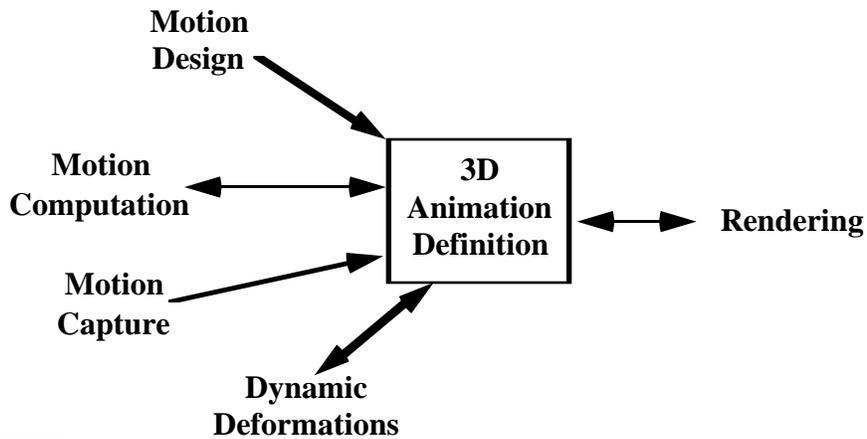
SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

The Graphics Process: Geometric Modeling



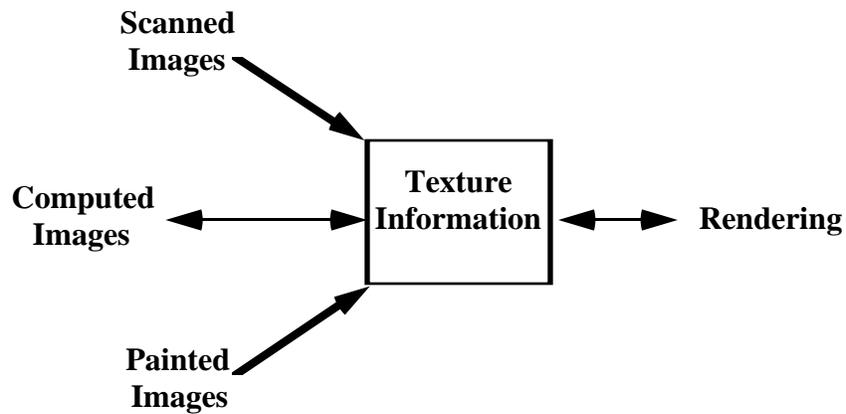
SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

The Graphics Process: 3D Animation



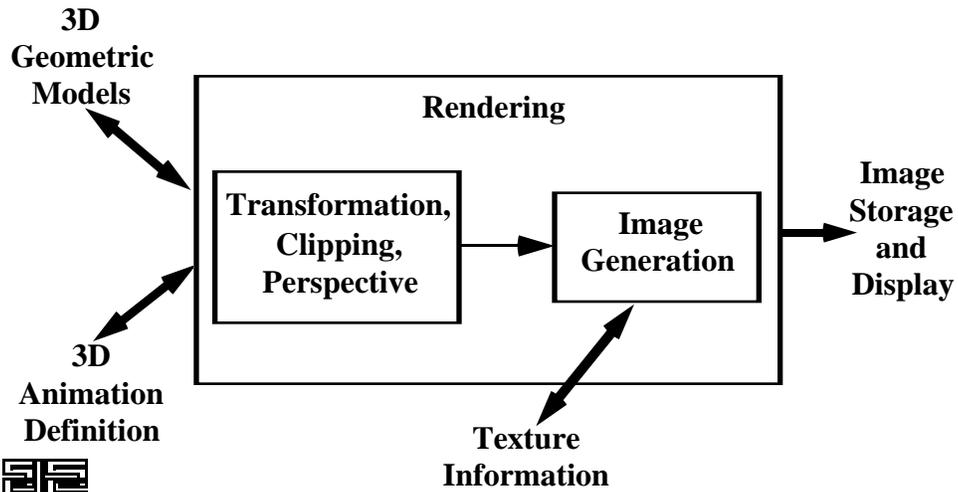
SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

The Graphics Process: Texturing



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

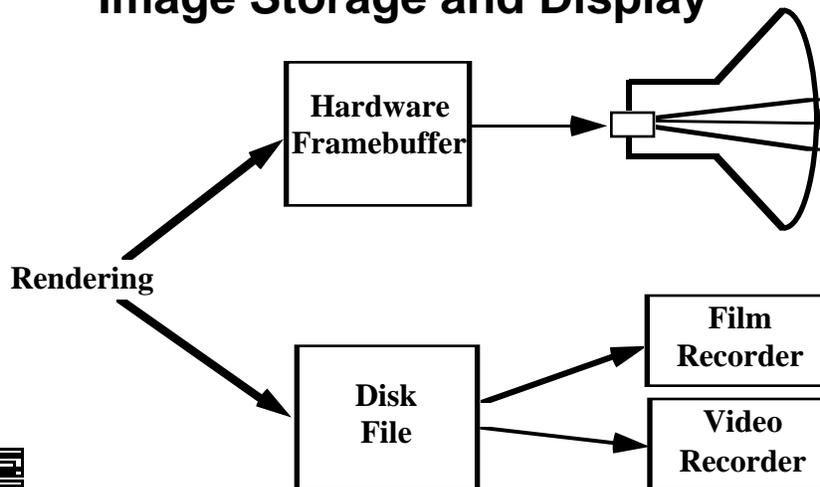
The Graphics Process: Rendering



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

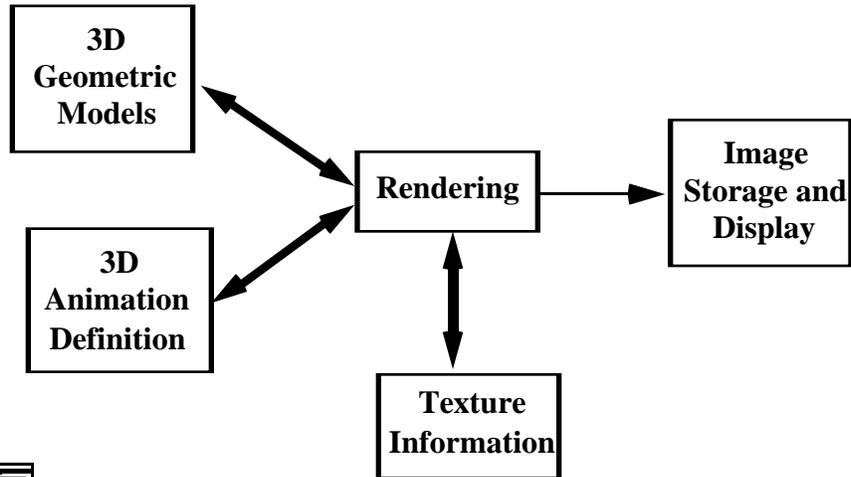
The Graphics Process: Image Storage and Display



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

The Graphics Process: Summary



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

An Introduction to Modeling

Andrew Glassner
Microsoft Research

Slide 1 of 100

Why Create 3D Models?

- **Image Synthesis**
- **Design**
- **Manufacturing**
- **Simulation**
- **Art**

Models for Image Synthesis

- **Camera**
 - Viewpoint for image
 - **Light Sources**
 - Radiate light
 - Have size and shape
 - **Objects**
 - Physical structures
-

Models for Simulation

- **Physics**
 - An airplane wing
 - **Mechanics**
 - Fit between parts
 - Manufacturability
-

Model Attributes

- **Structure**
 - Geometry and Topology
 - **Appearance**
 - Looks and surfaces
-

Levels of Detail

- **Visual detail for images**
 - **Structural detail for simulation**
-

Photograph

Telephone handset cord

Detail for Image Synthesis

- **Real shapes are complex!**
 - **More detail = more realism**
 - Takes longer to model, longer to render, and occupies more disk space
 - **Procedural objects**
 - More detail when you want it
-

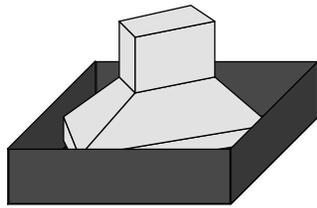
Photograph

Object in a room, in a far corner (low detail version) and close up (high detail version)

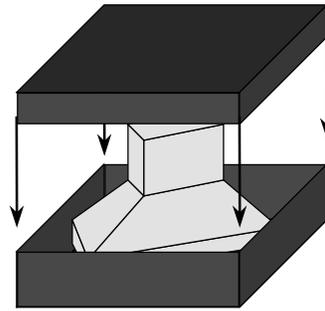
Detail for Simulation

- **Can affect accuracy of simulation**
 - **Different simulations require detail in different places**
-

Levels of Detail for Simulations



Does it fit in the box?



Does it fit with the cover on?



Photograph

Closeup of a stapler where the staple folds (high detail) and just checking page clearance (low detail)



Types of Modelers

- **Interactive**
 - **Scripted**
 - **Data Collection**
 - **Others**
-

Primitives and Instances

- **Platonic “ideal”**
 - **Shapes are instances of primitives**
 - **Each instance may be different**
-

Choosing a Model Representation

- Cost
 - Effectiveness
 - Complexity
 - Ease of Simulation
 - Ease of Animation
-

Model Cost

- Designer's time
 - Computer Storage
 - Rendering Time
 - Simulation Time
 - Ease of Animation
-

Model Effectiveness

- **Geometry**
 - Looks
 - Accuracy
 - **Appearance**
 - Looks
 - Accuracy
-

Model Complexity

- **Number of primitives**
 - **Number of shapes**
 - **Complexity of each instance**
-

Model Simulation

- **Is shape matched to simulator?**
 - **Cost of conversion**
 - Time and storage
 - Maintaining duplicate versions
-

Model Animation

- **Articulation**
 - Getting at the part you want
 - Getting it to move correctly
 - **Physics of motion**
 - **Constraints**
-

Modeling and Rendering

- **Rendering adds light to the model**
 - **The renderer tracks the light in the scene to determine how the scene looks to the viewer.**
 - **Lights and cameras are part of the model.**
-

Modeling and Animation

- **Animating is giving variations on a model over time**
 - **Different *keys* given by the animator are interpolated to give *in-betweens***
-

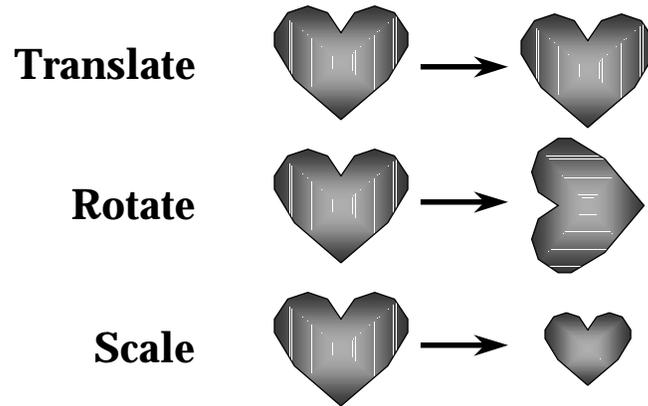
Levels of Detail

- **Match complexity of model to its use**
 - **Switch levels of detail**
 - Requires multiple copies of each model
 - Labor-intensive
 - Switching is hard to hide
-

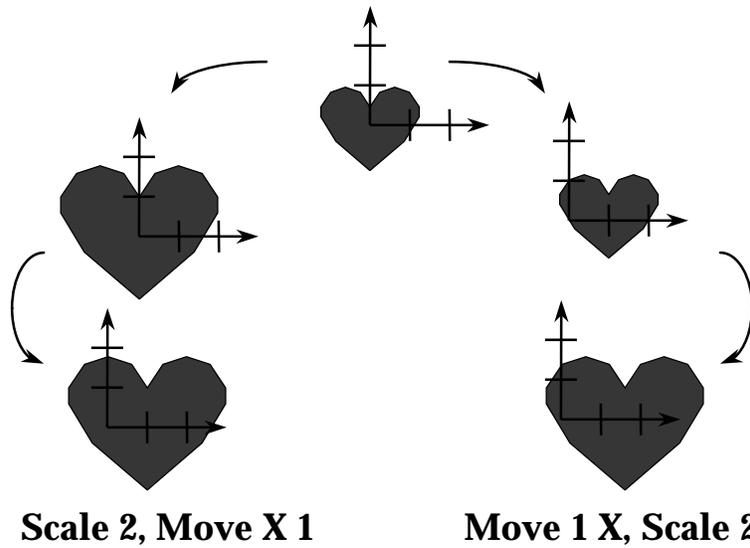
Procedural Models

- **Create model on demand**
 - **Models from coarse to fine**
 - **Switching is still hard**
 - **Requires skillful programming**
 - **Still an open research problem**
-

Basic Linear Operations on Primitives



Operation Order Matters!



Free-Form Deformation

- **Change the space, not the object**
 - **Great for animation**
 - **Allows flexible transformations**
 - **Bend**
 - **Twist**
 - **Taper**
 - **Melt**
 - **Etc.**
-

Photograph

Example of free-form deformation

Types of Primitives

- **0 Dimensions: Points**
 - **1 Dimension: Lines**
 - **2 Dimensions: Surfaces**
 - **3 Dimensions: Volumes**
-

Point Primitives

- **Particle systems**
 - **Requires many particles**
 - **Often procedurally controlled**
-

 **Photograph**

Example of a point-based model

 **Photograph**

**Drawing toy for making faces by
moving around magnetic filings.**

Surface Primitives

- **Polygons**
 - **Patches**
-

Polygons

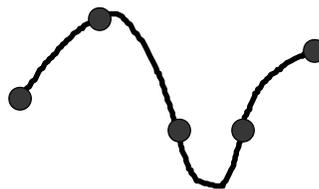
- **Simple to define and use**
 - **Assemble to make polyhedra**
 - **Flat**
 - **Flat**
 - **Flat**
 - **Really, really flat, always**
-

Patches

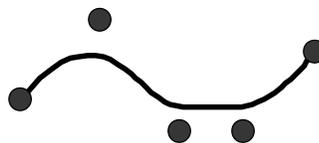
- Naturally curved
 - Defined by control points or curves
 - Interpolating
 - Approximating
-

Interpolation and Approximation

Interpolation



Approximation

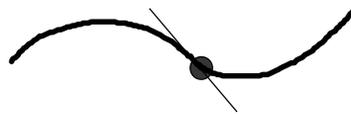


Continuity

0-order



1-order



2-order



Types of Patches

- Bezier
- B-spline
- Bicubic
- NURBS
- many more

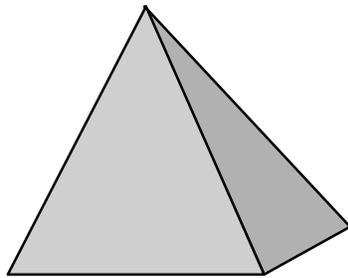
Volumetric Primitives

- **Volumes that enclose some space**
 - **Open vs. closed**
 - **Can be complex, e.g. a donut**
-

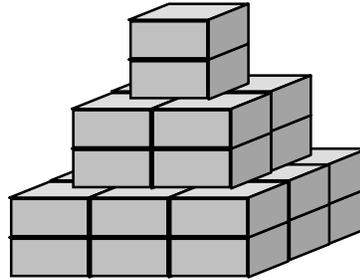
Voxels

- **Small blocks of space**
 - **Equally-sized (grid)**
 - **Varying sizes (octree)**
-

Voxels for Approximation



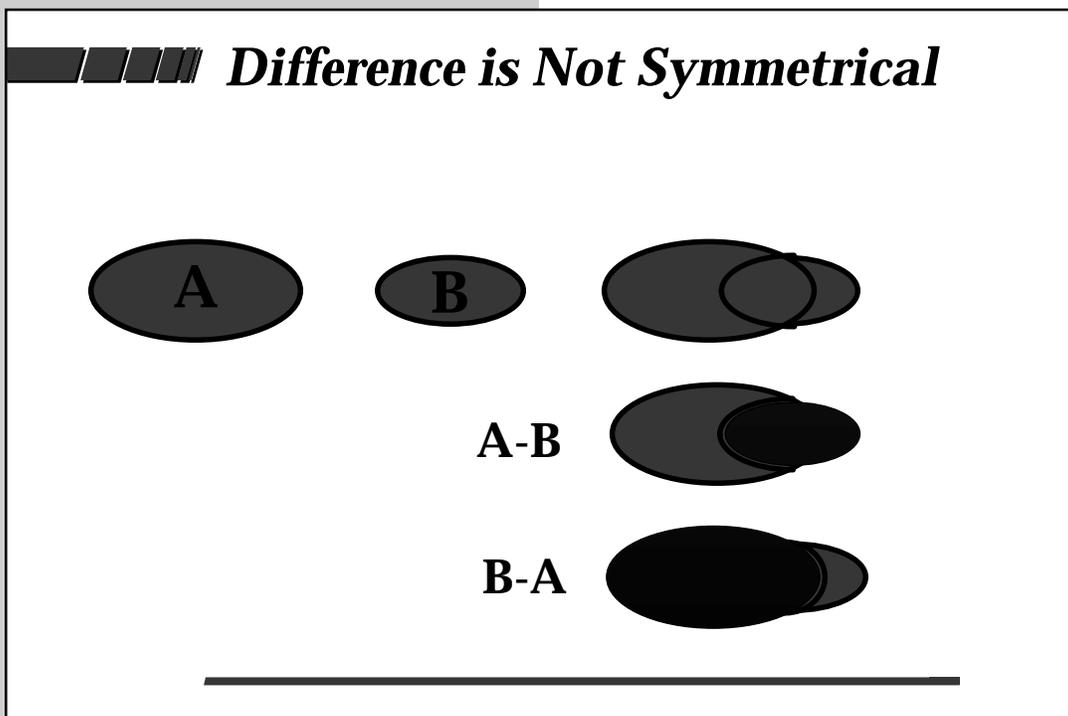
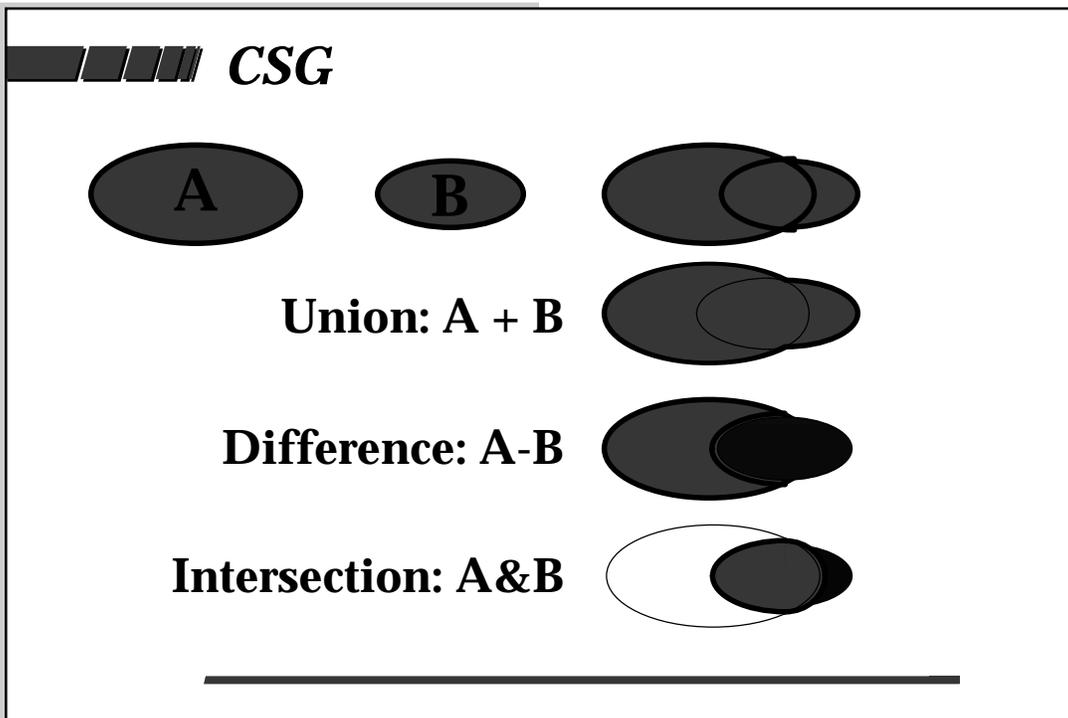
Original Shape



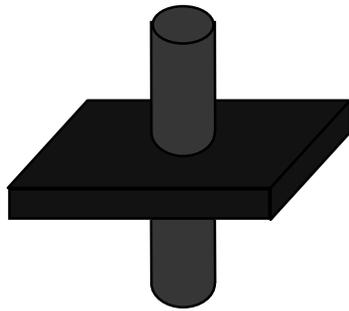
Voxel Approximation

Constructive Solid Geometry (CSG)

- **Combination rules for solids**
- **Each combines two solids**
- **Results can be used as a new solid**
 - CSG Tree
- **Three (or four) rules**



Difference is Useful for Cutting Holes



Block + Cylinder

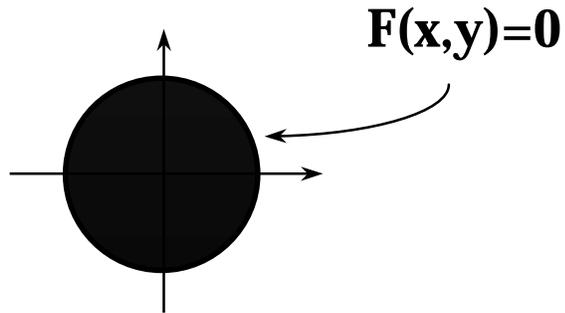


Block - Cylinder

Fillets and Blends

- **Make a smooth join between surfaces**
 - **Hard to do automatically**
-

Algebraic Functions



$$F(x,y) = x^2 + y^2 - \text{radius}^2$$

Blobs

- Algebraic functions, usually spherical
 - Add together to make smooth blends
-

Photograph

Example of a blobby model

Procedural Models

- **Fractals**
 - **Graptals**
 - **Shape Grammars**
 - **General Procedural Models**
-

Fractals

- **Self-similar**
 - **Infinite detail**
 - **Computer only approximates**
 - **Difficult to control**
 - **Mountains and ferns**
-

Photograph

A fractal model

Graptals

- **Make the structure first**
 - **Add geometry later**
 - **Useful for plants and organic forms**
 - **Data Amplification**
-

Photograph

A graptal model

Shape Grammars

- **Shapes that turn into other shapes**
 - **Details that work with substrate**
 - **Data Amplification**
-

Photograph

A model made with a shape grammar

General Procedural Models

- **Most powerful technique of all**
 - **Can handle smooth changes in detail**
 - **Supports simulation, animation**
 - **Allows models that interact with the scene**
-

Modeling Environments

- **Interactive**
 - **Scripted**
 - **Captured**
 - **Clip Art**
-

Interactive Modeling

- **Interactive**
 - **Exploratory**
 - **Immediate Feedback**
-

Photograph

An interactive modeling program

Scripted Modeling

- **Precise**
 - **Repeatable**
-

Photograph

A scripted modeler and script

Captured Modeling

- **Allows capturing real-world shapes**
 - **Generates rich models**
 - **Can be noisy**
 - **Can have geometric restrictions**
-

Photograph

A 3D scanner

Clip Art

- **Fast acquisition**
 - **Can be cheaper**
 - **May not be articulated as you want**
 - **Difficult to customize**
-

Modeling for Animation

- **Rigid structures are easiest to make**
 - **Articulated structures are easiest to animate**
 - **Plan for where you want motion**
 - **Built-in constraints**
-



Conclusions

- **Many primitives**
 - **Many modelers**
 - **Match your choice of modeler, primitive, construction style, and level of detail to the expected use.**
 - **Think before you model!**
-

3D Object Modeling

Norman I. Badler ^{*} Andrew S. Glassner [†]

1 What is 3D modeling?

Every computer-rendered image requires three essential components: a 3D scene description, one or more sources of light, and a description of the camera or eye viewing the scene. The scene description is typically composed of one or more *models*, or 3D structures. Typically we think of a *model* as a stand-alone part, e.g. a pencil or a tree, and the *scene* as the assembly of these parts into a complete 3D environment. This attitude reflects the most common procedure for building up a 3D scene: one builds many models, and then assembles them.

Each model contains two descriptions: a mathematical representation of the structure of the shape, and a recipe for figuring out how that shape would look if illuminated. Let us examine these descriptions in turn.

1.1 Model Structure

The structural description is basically one of *geometry*. It tells us where the object is in space, and where it is not. Imagine an empty coffee mug hovering in space in front of you. Now freeze time and look at every molecule in the room: generally each molecule will be part of the material of the mug or part of the air around the mug (note that the air molecules inside the mug where the coffee would go are not part of the mug itself; they're part of the air around it). If you paint every air molecule white and every mug molecule black, then you'll have a very fine description of the mug down to the precision of the molecules in the room.

The result of this thought experiment has many of the properties of 3D models used in computer graphics: it's conceptually straightforward (if bulky), and has a limited precision (most computer programs use the built-in calculation hardware in today's computers; this hardware has very high but limited precision). Some modeling methods are very close to this approach; they create points in space or chop up space very finely and label it empty or full. Other

^{*}Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA 19104 USA

[†]Microsoft Research, One Microsoft Way, Redmond, WA 98052

methods attempt to let you describe your model more abstractly, saying for example that your mug is basically a thick cylinder with a bottom and rounded lip.

1.2 Model Appearance

The other part of every model is the *surface description*. This boils down to describing the *physics* of how the surface of the model interacts with light. Happily, it turns out that these physics are driven by a few descriptive terms that have intuitive meaning for us, such as color, shininess, and transparency. Thus you could say that your coffee mug is shiny and blue, and go a long way towards describing its appearance.

Since models often simulate real world shapes, we have to decide how detailed we want to be. When describing the coffee mug's geometry, do we want to include every little bump in the ceramic and the irregularities in the overall shape, or is a mathematically perfect cylinder close enough? When describing its appearance, do we want to include the little scratches in the glaze, or is it enough to say that it is just uniformly shiny? The answers to these questions depend on how you intend to use the model. If it's going to be sitting on the shelf in the background of some scene, poorly lit and barely visible, then a blue cylinder might do the trick. But if the mug is the star of your image, front and center, large and well-lit, then you will probably want more detail.

2 Models: Images and Simulations

There are two principle reasons for building 3D models on the computer: image-making and simulation. These two procedures are very similar, but they typically have different goals.

Image-making requires models that *look good*. They need to be sufficiently complex that they aren't boring to the eye, and sufficiently realistic (when that's the goal) to pass convincingly as a good description of the real shape. The ultimate arbiter of the model's quality is the person who looks at the picture.

Simulation requires models that are *accurate*. Some simulations test two models mathematically to see if they fit together within some tolerance; the result is simply a yes or no answer, rather than an image. Consider a simulation that tests the strength of the roof of a new stadium under conditions of heavy rain: it's critical that the simulated roof have exactly the intended shape in order to determine how much rain will roll off the sides. An airplane wing also must be modeled with high accuracy if we expect to learn anything meaningful about its lift characteristics when actually actually built.

Some models are the result of a simulation: given a computer-controlled drill, a block of wood, and a program for the drill, what is the shape of the resulting part? Here we only supply the initial model, and the computer modifies the

model for us. We may be as interested in the surface finish of the final piece as its geometry. A hole might be round, for example, but we would like to be sure that it is smooth.

These two applications have spawned two rather different ways of describing models. One is interactive and approximate, appropriate for making models that just need to look good. The other is slower and precise, appropriate for models that must be accurate. Of course, there's a huge overlap between these different systems, since many models used for images need to be very precise, and sometimes simulations just need something close as a starting point or stand-in for a more complex shape.

3 Detail: How Much?

Details add interest. The amount of detail in a model can make the difference between a boring and fake image, or one that is interesting and realistic. For simulation, details can make the difference between getting an answer that is right or wrong.

3.1 Visual Detail

Early computer graphics images used very simple models: a table was a long box, made up of six perfectly flat, perfectly smooth, perfectly colored sides. Real tables are much more interesting because they have details. They have structural detail, and appearance detail (also called *texture*). Look at any man-made or natural 3D object, and you'll see a wealth of shape detail and appearance detail: nothing is perfectly flat, smooth, or featureless.

To make interesting and believable images, we need models that have enough richness that they begin to match the complexity of the real world (idealized forms are useful for art and education, but even there complexity can help you get your point across). The complexity of what seem to be simple forms can be staggering: look closely at the coiled line running from a telephone handset to the base. The wires follow a very complicated path, the gaps between coils increase and decrease, the coils themselves loop around and through each other. The finish on the wire isn't consistent either: there are scratches and discolorations all along its length, sometimes in random places and sometimes in organized chunks (like where the cord got caught and scraped in a doorway one day). It's this kind of detail that makes a model interesting, and creating this detail is one of the principle challenges to anyone who makes models.

In general, more detail in the geometry and the appearance is always better. But there are two caveats to this principle: more detail takes you (the *designer*) more time to create, and it takes the computer (the *renderer*) more time to process. In the previous example, a very detailed mug sitting in a dark corner will just slow down your picture-making and add nothing. The situations that

cause the most trouble occur in animations, when a shape starts out far away and barely visible, and later comes to occupy much of the image (such as a baseball hurtling towards the camera).

3.2 Simulation Detail

If you're building a model of a stapler and you want to make sure that the staples will be delivered correctly, it will be important to be very accurate in your model of the little hollow depression in the base where the staple folds. But if you only want to make sure that a pile of paper 30 pages thick will fit in the opening between the head and the base, the exact shape of the hollow won't matter. Finding the right level of detail is a matter of matching purpose to effort.

4 Modeling Software

There are several types of programs available to help you create 3D models. These programs are usually referred to as *modelers*. The person using the program is also sometimes referred to as the modeler, but that can be confusing (and dehumanizing). It seems more personal to think of the person as the *designer* (this is also more meaningful than the generic and uncomplementary *user*).

One idea that is common to most modelers is that the designer can create a copy of some basic form (such as a block or patch), and then modify it to create part of the model. The basic forms are called the *primitives* for that modeler, and each copy of is an *instance* of that primitive. So a blocky robot made of 20 rectangular blocks contains 20 instances of the primordial block. This approach is reminiscent of Plato's notion of ideal forms; the prototype shape is the ideal, and instances are practical realizations of that ideal.

4.1 Interactive Modelers

One class of modeler is *interactive*. Typically the designer sits in front of a terminal with a keyboard and mouse, and manipulates shapes on the screen. Because the image is only two-dimensional, different systems have different conventions to allow you to manipulate objects in 3D.

Interactive modelers typically contain a variety of design aids to help you achieve some precision in your model: a common example is to "snap" points on the model to other model points or to an invisible grid.

Models are built by interactively selecting, creating, modifying, and assembling instances of primitives. The shapes typically have "handles" that allow you to control the shape. Handles may come in the form of points, curves, or interactive tools.

Often one determines the surface characteristics when the shape is first created, but then textures may be laid on the surface to modify it.

4.2 Scripted Modelers

Another class of modeler is relies on an input *script* to define the model. This is typically a text file that has been created by the designer using any conventional text editor. The script specifies the shapes in the model one by one, identifying each one by its primitive, and the parameters that specify that instance.

Because scripted modelers allow you to specify operations numerically rather than interactively, they are ideal for very precise modeling. For example, it might be difficult to interactively place one block at a 37.5° angle to another, but it's trivial to specify when you can type that angle in.

4.3 Other Modelers

There are other ways to build models, for example by digitizing existing 3D structures, or by analyzing photographs. We will return to these techniques near the end of this paper.

5 How are models used?

There are many uses for 3D models. While there is an obvious desire to experience an object visually – hence computer graphics – there are other important motivations that form a larger and broader context for studying object models. Some of these uses (with examples) for 3D models are listed below. Note that each one tends to emphasize either the visual or simulation aspect of the model as discussed earlier.

- To visualize designs: How does a product look before it is manufactured? What is the best combination of shape, color, layout, etc.?
- To assess appearance: How does some existing environment look if changes are made to it? Does it need more or less lighting? Are the vistas appropriate?
- To observe part relationships: How do things fit together? What does an exploded view look like? Do parts touch or collide that should be separated?
- To check feasibility (of manufacture): Does the object meet the design specifications? Can it satisfy constraints on stress, loading, heat transfer, etc.? Can it be effectively machined, cast, or sculpted?

- To determine cost, volume, area, machining time, etc.: How much material is needed? How much material must be removed by machining? What is the cross-sectional shape of various internal structures in a complicated assembly?
- To determine faithfulness to physical phenomena: How does the surface of the object interact with light? How do the surface properties (smoothness, roughness, waves, etc.) depend on the geometry of the object? How close in appearance to “reality” can we get?
- To exercise display algorithms: What limits might we have on the computations over the object models? What determines graphical algorithm complexity? What display algorithms are needed to create images of a particular kind of model?
- To express artistic goals: What juxtaposition of the real and imaginary objects will portray an artists’ visual message and mood? How much can the imagination be stretched to visualize artificial worlds?

6 The major issues

The major issues involved in object modeling include the computational cost of the model, its effectiveness in modeling the desired phenomena, its implementation complexity, and the methods used to acquire (or create) data that describe the object geometry.

The computational cost of a model may be gauged in terms of computer storage space, object construction time, display time, or in application use. For example, the storage space of a model may be based on the space required to store each primitive and the number of primitives needed to represent it. Thus a sphere is a relatively simple primitive; collections of spheres grow linearly with the number of spheres in the model. Polygon models may require variable length storage for an (arbitrary sided) polygon, plus additional link structures to organize a set of polygons into a closed surface.

In object construction time, the costs may be based on where the data is and how the data is captured. Clearly more automated data capture can favor one object model over another that requires considerable manual effort. Once the data defining the model is available, variations from the original input should be possible to clean up noise introduced by automated methods, vary shape parameters obtaining similar but individual members of an object class, or simply correct designer input errors.

Computer graphics has frequently focused on the algorithmic side of the display task. There are costs associated with the display algorithms used to visualize each type of object model. Visualization cost can be as low as that of simple linear traversal and wireframe display of polyhedral objects, or as

costly and complex as ray-traced shaded renderings of algebraic surfaces. The fundamental observation is that display cost is dependent on the object model; computer graphics history is, in a large part, the study of procedures to display a particular class of object models. As in many other computer science algorithms, one pays in time for increasing generality of function. Thus the cost of ray tracing also entails the power to display a wide variety of object model types; at the other end of the spectrum, a simple depth buffer algorithm is highly effective for real-time display provided one is restricted to polyhedral models. Finally, the display cost is also dependent on image quality issues; for example, high resolution or animated images will require better models (and usually more detailed models), visualization studies might require particular object surface properties for advanced lighting models, etc.

Models are used for other purposes besides display, however, and the relative costs and complexity of these used must also be considered when selecting a model type. Two common uses are the transformation of an object in position, size, or shape, and the measurement of object geometric properties such as volume. These uses have costs associated with the geometric algorithms but the standard complexity measures must be weighed against the expected use. For example, the more efficient computational algorithm in the general case may have constants which make it less satisfactory in the expected majority of cases: a model consisting of arbitrary polygons will have less scan conversion efficiency than one composed entirely of triangles. Also, a frequently repeated algorithm should be made more efficient than one used once. Thus a simpler but theoretically costlier algorithm may be more efficient when multiplied by expected use. When programmer time is factored in, the costs of a graphical algorithm become a complex mix of special case considerations, overall efficiency, and programmer efficiency.

Certainly one important issue in object modeling is effectiveness: whether the model actually represents the desired phenomena and provides computational analogs of the appropriate visual or physical properties. Poor visual quality may be (though is not identical to) poor modeling. Thus the visual effectiveness of a stick figure of a human is compromised by the zero thickness model of body segments. On the other hand, a fractal model of a mountain may not tell us anything about the formation process of real mountains, but the visual effect and the statistical properties of the resulting surface may be quite convincing.

The complexity of a model may be judged better as a trade-off between the number of primitives and the inherent complexity (without being circular in reasoning) of the individual primitives. Thus at one end of the scale are objects which can be characterized as a multiplicity of simple forms; at the other end are objects which seem inherently complex. For example, machine parts, buildings, and other man-made objects seem to be mostly a multiplicity of simple forms, while biological and other natural structures (mountains, textures) are inherently complex.

Often object data will be available naturally in one form yet needed in another. This requires conversion from one form to another. The input may be formatted by an available input device (digitizer, laser scanner, tomographic images, manual drawing input, etc.), but the requirements of the application may dictate conversion to another form for more convenient display or computation. For example, surface point data may be approximated by polygons or curved surfaces to permit visual surface rendering in workstation hardware; constructive solid geometry models might be converted to voxels for volume computation.

7 Models and Rendering

It is important to understand the difference between models and rendering. Models describe the object and its attributes such as shape or geometry, color, reflectivity, transmittance, surface smoothness, and texture. Shade trees may be used to ascribe different visualization algorithms to different parts of a model. But it is the rendering algorithm itself which transforms the model to a screen-based view from a given camera position, projects the 3D data into display coordinates, determines the visible portions of the scene, and converts the object properties into pixel values in the context of light sources, atmospheric effects, anti-aliasing, image compositing, and color models.

8 Operations on models

We will ignore most of the rendering issues as they are discussed elsewhere in this Tutorial. We proceed to discuss the operations appropriate to object models. These include transformations, change of detail, measurement, combination, deformation, and display.

Transformations form the basis of most modeling systems. These include the well-known geometric transformations of translation, rotation, dilation (scaling), and reflection.

Change of detail is the process of adding or removing information to produce a finer grained representation (for better visual effect) or simplify other operations (for computational efficiency). Usually some interpolation method is used to augment detail: for example, fitting curved surface patches to point data to produce smooth continuous approximations to the expected original (partly-sampled) data. On the other hand, averaging or subsampling of the data may be used to reduce detail. Often models are grouped into hierarchic structures to simplify their organization: a model consists of sub-models, each of which consists of sub-models, etc., down to the level of one or more primitive types. An alternative hierarchic structure may be formed by defining the sub-models to be the same object as the parent object, only at a lesser degree of

detail. During model rendering, the appropriately detailed model is selected for display based on the projected size of the entire model on the display surface. During animation, as the size changes, the models from two detail levels may be visually blended with each other.

A number of measurements may be made on object models. The topology may be computed to determine whether the model is totally connected or consists of separate components. The presence of topological holes can be determined. The overall consistency of the object may be tested; for example, in surface-based models it is desirable to utilize topologically well-defined Euler operators to insure the validity of the object models being interactively designed.

Other useful or common operations include point-to-point distance measurement, computation of surface area or volume, and determination of surface tangents or normals for the rendering process.

Object models may often be combined to form new and more complex objects. The constructive solid geometry system is in fact based on Boolean combination as the fundamental operation. The combination operations are union (everything that is in either object), intersection (everything that is in both objects), difference (everything that is in one object but not the other), and symmetric difference (everything that is in either object but not in both). Other combination operations are the cut or slice which exposes the interior to view, and the cover operation which defines the relationship of one object to another in the finished image. The covering is an image property which does not necessarily stem from a real 3D model motivation.

There are a number of ways in which an object may be deformed. Deformation enriches the selection of object models without resorting to generation of totally new data sets. Deformation operations include skew, stretch, fold, and perturb (e.g. randomly, stochastically, or fractally).

The display operation is the most visible operation on an object model. There are many rendering algorithms, many of which are discussed elsewhere in this Tutorial: wire-frame, visible line, visible surface, ray casting, and ray tracing.

9 Representation structures

As we remarked earlier, the representation structures used for an object model may be either declarative or procedural. In a declarative representation, the model is explicitly embedded in a standard computational data structure. In a procedural scheme, the model is embedded into any convenient computational procedure, such as a formula, implicit equation, or arbitrary code. In the former, data is retrieved by search, indexing, or pointer chasing; in the latter, data is obtained by invoking a procedure with passed parameters or sending a message to an object which then executes a response.

10 Model classification

The major concern of the rest of this discussion is a taxonomy of models. We classify models into two broad categories: boundary schemes and volumetric schemes. In a boundary representation the surface of the object is approximated by or partitioned into (non-overlapping) 0-, 1-, or 2-dimensional primitives. We will examine in turn representations consisting of primitives formed by points, implicit surfaces or algebraic equations, polygons, fractals and graftals, and curved surface patches. In a volumetric representation the 3D volume of the object is decomposed into (possibly overlapping) primitive volumes. Under volumetric schemes we discuss voxels, octrees, constructive solid geometry, specialized (single primitive) systems, potential functions, and particle systems.

10.1 Surface and boundary models

The simplest surface model is just a collection of 3D points. They may be organized in a simple list, or may be more highly structured as contours, slices, or sections. Surfaces represented by points require a fairly dense distribution of points for accurate modeling.

10.2 Implicit surfaces / Algebraic equations

These surfaces are defined as the solutions to algebraic formulas. One familiar class of such surfaces are the quadrics. In general, the surface is the solution to an equation such as $F(x, y, z) = 0$ and numerical techniques or search are required if F is not analytic.

10.3 Polygons

Polygonal (polyhedral) models are one of the most commonly encountered representations in computer graphics. The models are defined as networks of polygons forming 3D polyhedra. Each polygon (primitive) consists of some connected vertex, edge, and face structure. There are a variety of data structures possible. Groups of polygons are stored in lists, tables, or linked structures to facilitate traversal of connected faces, the edge network, etc. The polygons are sized, shaped, and positioned so that they completely tile the required surface at some resolution. Polygon models are relatively simple to define, manipulate, and display. They are the commonest model processed by workstation hardware and commercial graphics software. In general polygons are best at modeling objects meant to have flat surfaces, though with a large enough number of polygons quite intricate and complex objects can be represented. “Large enough“ may mean hundreds of thousands of polygons!

10.4 Fractals and Graftals

A limitation of the surface point and polyhedral models is the necessity (usually) of explicitly storing all the requisite data. If the objects to be modeled are highly irregular or complex, such as mountains or clouds, their statistical shape properties may be used to better advantage. Fractals and graftals create surfaces via an implicit model that produces data when requested. The surfaces are frequently based on point or polygonal models that are decomposed into finer and finer detail as the requirements of the display dictate. Fractals use special “random” processes to achieve a naturally motivated statistical irregularity such as demonstrated in rocks, mountains, clouds, coastlines, etc. Graftals use deterministic processes to model more repetitive patterns such as trees, leaves, snowflakes, etc. True fractals have a self-similarity property, where the frequency distribution of shape is supposed to be the same no matter what scale is chosen for examining the object. In practice, there are several variants on the true fractal definition.

10.5 Curved surfaces

Since polygons are good at representing flat surfaces, considerable effort has been expended determining mathematical formulations for true curved surfaces. Although implicit surfaces may exhibit true and continuous curvature, the need to solve equations is costly and usually undesirable. Most curved surface object models are formed by one or more parametric functions of two variables (bivariate functions). Each curved surface is called a patch; patches may be joined along their boundary edges into more complex surfaces. Usually patches are defined by low order polynomials (typically cubics) giving the patch easily computed mathematical properties such as well-defined surface normals and tangents, and computable continuity conditions between edge-adjacent patches. The shape of a patch is derived from control points or tangent vectors; there are both approximating and interpolating types. The former take the approximate shape of the control vertices; the latter must pass through them. There are numerous formulations of curved surfaces, including: Bezier, Hermite, bicubic, B-spline, Beta-spline, polynomial, rational polynomial, cardinal splines, composite splines, etc.

10.6 Volume and CSG models

The first volumetric model we examine is the voxel model. Here space is completely filled by a tessellation of cubes or parallelepipeds called voxels (volume elements). Usually there is a density or other numerical value associated with each voxel. Storing a high resolution tessellation is expensive in space but simple in data structure (just a large 3D array of values). Usually some storage optimization schemes are required for detailed work (1K x 1K x 1K spaces). Special

techniques are needed to compute surface normals and shading to suppress the boxiness of the raw voxel primitive. Voxel data is commonly obtained in the medical domain; it is highly regarded for diagnostic purposes as the 3D model does not speculate on additional data (say by surface fitting) nor suppress any of the original data however convoluted.

10.7 Octrees

Octrees are one of the data structures used for volumetric models that tessellate a given 3D space. The original volume, say a cube, is partitioned into 8 cubes if it is non-empty. Recursively, each sub-cube is partitioned whenever non-empty, until some minimum size element is reached. Since empty cubes are not sub-divided, the storage space efficiency is increased. The major use of octrees appears to be an indexing scheme for access efficiency in a large 3D array.

10.8 Constructive solid geometry

One of the most efficient and powerful modeling techniques is constructive solid geometry. Unlike the voxel and octree schemes, there is no requirement to regularly tessellate the entire space. Moreover, the primitive objects are not limited to (uniform) cubes; rather there are any number of simple primitives such as cube, sphere, cylinder, cone, half-space, etc. Each primitive is transformed or deformed and positioned in space. Combinations of primitives or of previously combined objects are created by the Boolean operations. An object therefore exists as a tree structure which is “evaluated” during rendering or measurement.

10.9 Specialized (single primitive) systems

The generality of the constructive solid geometry method – with its multiplicity of primitive objects and expensive and slow ray-tracing display method – is frequently reduced to gain efficiency in model construction, avoid Boolean combinations other than union, and increase display speed. The idea is to restrict primitives to one type then design manipulation and display algorithms to take advantage of the uniformity of the representation. Voxels might be considered such a special case, where the primitives are all coordinate axis aligned and integrally positioned cubes. Other schemes are possible, for example, using ellipsoids, cylinders, superquadrics, or spheres.

Ellipsoids have been used to model cartoon-like figures. They are good for elongated, symmetric, rounded objects. Unfortunately, the display algorithm is nearly the same as the general ray-tracing process. One unusual application of (overlapping) ellipsoids is to model terrain, clouds, shrubbery, and trees. In these cases, the convincing visual effect of semi-solidity is achieved by statistical transparency which has a low rendering cost.

Cylinders have also been used to model elongated, symmetric objects. An application that uses cylinders plus spheres with a special-purpose display algorithm is ball and stick molecular modeling.

Superquadrics are a mathematical generalization of spheres which include an interesting class of shapes within a single framework: spheres, ellipsoids, and objects which arbitrarily closely look like prisms, cylinders, and stars. Simple parameters control the shape so that deformations through members of the class are simple and natural. Superquadrics are used to model man-made objects, but when overlapped can give the appearance of faces and figures.

Spheres as a single primitive form an intriguing class. They are the only modeling primitive that is isotropic: that is, identical in appearance from any view. Moreover, spheres have a simplicity of geometry that rivals that of simple points: just add a radius. There are two methods of rendering spheres. They can be drawn as fully 3D objects in which case some scan conversion tricks can be used to simplify the generation of successive object points (basically a generalization of incremental circle drawing algorithms). An alternative display method treats the spheres as if they were “scales” on the modeled object; in this case a sphere is rendered as a flat shaded disk. With sufficient density of overlapping spheres, the result is a smoothly shaded solid which models curved volumes rather well. A naturalistic human figure may be done this way.

10.10 Potential functions

An interesting generalization of the sphere model is to consider the volume as a potential function with a center and a field function that decreases monotonically (by an exponential or polynomial function) from the center outward. There is no “radius” or size of the potential function; rather, the size or surface is determined by setting a threshold value for the field. What makes this more interesting is that potential functions act like energy sources: adjacent potential functions have overlapping fields and the resultant value at a point in space is in fact the sum of the fields active at that point. Thus adjacent fields blend smoothly, unlike the “creases” that are obtained with fixed radius spheres. Recently, directional dependence and selective field summation across models have been added to create “soft” models that blend with themselves but not with other modeled objects in the environment. Potential functions were originally used to model molecules, since atoms exhibit exactly this form of field behavior. Other uses for potential functions are real and imaginary organic forms including human and animal figures.

10.11 Particle systems

Generalizing spheres in a different direction, particle systems reduce the sphere to a zero radius. A volume is therefore characterized by a set of (usually moving) particles which indirectly define the space. Each particle has its own color,

path, history, and lifetime. Their motion is typically controlled by probabilistic algorithms. Particle systems have been used to model fire, gases, explosions, fireworks, and grassy fields.

11 Where do models come from?

There are numerous methods for generating 3D surface or volume models. We briefly examine manual digitization, semi-direct data acquisition, and algorithmic methods.

Manual digitization is the construction of a model from 3D coordinate data measured and typed in or directly traced from plans, models, or the real thing. Several forms of digitizers exist to aid this task. Often data from two or more views is used so that 2D data may be extrapolated into 3D. If two views are taken with properly positioned and calibrated cameras, manual selection of corresponding points on the two views can be input to a 3D reconstruction formula. This is the basic operation in photogrammetry.

If manual input is too tedious, or the model too complex, semi-automated data acquisition methods may be used. Principals among such methods are direct 3D coordinate input via 3D digitizers, laser scanning to obtain gridded range (hence depth) information, voxel reconstruction from multiple planar (tomographic) images, and direct (video) image analysis. The last method is still in relative infancy, but computer vision techniques can be used to intelligently reconstruct various simple geometric models based on polyhedra, curved surfaces, spheres, cylinders, and superquadrics.

Algorithmic and interactive design methods form a large class of model building techniques. Typically an interactive graphical editor is used to build a model within the particular modeling system representation. For example, polyhedral models are pre-eminent in computer-aided design applications, with curved surfaces and constructive solid geometry models becoming widely available. Other interactive model editors have been constructed for the other model types. Often, an algorithmic method can assist the design process by enforcing constraints on the model primitives. For example, using Euler operators will insure that a polyhedral model is topologically well-defined (though it may still be odd in a geometric sense). Actual constraint operators may insure that features within a model are parallel, tangent, horizontal, vertical, non-intersecting, etc. Finally, some algorithmic methods can be used to interpolate or generate data from forms more easily input; for example, by creating reconstructed surfaces between parallel contour slices of surface points, by fitting curved surfaces to control vertices or tangency constraints, or by using force or energy methods to cause model pieces to self-assemble.

A Glossary for Modeling and Animation

Norman I. Badler * Andrew S. Glassner †

Adaptive sampling Evaluating an image coarsely at first, and then more finely in regions with detail. Causes for taking more samples include complex shapes, textures, and shadows. See *super-sampling*.

Aliasing Artifacts in an image generated when we only use a finite number of equally-spaced samples, such as at pixel centers. The artifacts include banding, Moirè effects, jaggies on polygon edges, and regular patterns that don't belong.

Algebraic Surface A surface defined by an equation. The surface contains all points which evaluate to a particular value of the equation, usually zero. Algebraic surfaces are a class of implicit surfaces.

Ambient The amount of illumination in a scene which is assumed to come from any direction and is thus independent of the presence of objects, the viewer position, or actual light sources in the scene.

Animation The process of creating and recording images which change over time. Though often interpreted as implying only two-dimensional image changes, it may be applied in general to any model or scene changes in three dimensions as well.

Anti-aliasing The process of reducing, removing, or avoiding aliasing artifacts. See *aliasing*.

Articulation The ability of one part of an object to move with respect to another, such as a hinge or sliding joint.

Background A fixed image which is interpreted as existing infinitely far from the observer. It is displayed when no object surface falls into an image sample.

Bend Deforming a shape along a line.

* Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA 19104 USA

† Microsoft Research, One Microsoft Way, Redmond, WA 98052

Binary Space Partitioning (BSP) A method for dividing up a 3D space with a set of planes. For any observer and plane, each object in the environment either straddles the plane, is on the same side as the observer, or on the other side. Objects on the same side of (and in front of) the observer will block visibility of objects on the other side.

Bivariate patch A vector-valued function of two variables, typically called u and v . These parameters generate some locally two-dimensional surface in space. Often the parameters are assumed to lie in the range $[0,1]$. See **patch**.

Blob An implicit surface. Blobs are usually radially symmetric Gaussian functions.

Boolean operation An algebraic combination (union, intersection, or difference) of two geometric solids.

B-Rep See **Boundary representation**.

Boundary representation The geometry of an object as described by its surface. The surface of the object is approximated by or partitioned into (non-overlapping) 0-, 1-, or 2-dimensional primitives (points, curves, and surfaces).

BSP See **Binary Space Partitioning**.

B-Spline A type of spline, where the curve is influenced by the control points, but generally does not pass through them. See **Splines**.

Bump mapping A technique for faking fine-level geometry on a surface, such as the bumps on an orange rind. Rather than actually create many little pieces of surface to model the bumps, a texture map is placed on the surface which describes how the surface normal would vary if the shape really had bumps. By shading with these imaginary normals, the surface appears to be bumpy. Drawbacks to the technique are related to the fact that only small bumps can be faked this way: the silhouette is not changed, bumps don't occlude other bumps, and usually the bumps don't cast shadows.

CAD Computer-Aided Design. The process of manually creating geometric object models which meet desired design criteria.

Catmull-Rom Spline A type of spline, where the curve is influenced by the control points, but generally does not pass through them. See **Splines**.

Cel animation Animation produced by stacking multiple 2D drawings (called *cels*), each containing a fragment of a scene or character on a transparent background. In a physical environment the cels are lit and photographed

to create one frame of animation. Cels are basically a labor-saving device. If a character is standing still and talking, then a cel of the character without a mouth might be placed over a background. Then for each frame a cel containing the appropriate mouth image is placed on top of the stack, avoiding the necessity to re-draw the entire character for every frame. Often there are several independent planes of images which may be superimposed in a so-called "two and a half-dimensional" animation.

Clipping planes Boundary planes in the world coordinate space which delimit the portion of the geometric model which will be rendered. Typically top, bottom, and side clipping planes are based on the window and camera position. The front and back (or *hither* and *yon*) clipping planes are used to select the object depth limits to remove undesired foreground (closer) or background (further) objects or slice the object perpendicular to the view to expose inner detail.

Compositing The process of creating an image by combining multiple independent images. A *matte* describes how much each point of each image contributes to the final picture. To animate a spaceship, for example, one might render the spaceship, and then composite it over a background of stars. The ship would completely replace the starfield, but some of the stars might be visible through the exhaust. In animation, the background and stationary objects are rendered once, while changing or moving objects are rendered on a per frame basis and combined with the static part of the image.

Constraints Values, relationships or conditions which are to be maintained while other changes are being made. For example, a constraint can hold a line horizontal, keep two objects a constant distance from one another as they move, or can cause one object to be attracted to another.

Constructive Solid Geometry (CSG) A model representation based on a set of primitive geometric shapes which may be transformed spatially and combined in a tree-structured fashion by Boolean operations. The resulting model is a tree with the primitives as the leaves, Boolean operations as the non-leaf nodes, and edges as transformations.

Contour Typically a curve in space. Common contours are those used as the cross-section for *swept surfaces*, and the silhouette of some object from a particular point of view.

Control Point One of the points that influences the shape of a spline or patch. See *splines*, *patches*.

Concave See *Convex*.

Continuity When two patches share an edge, that edge may be more or less noticeable. The *continuity* of the edge across the patches is a mathematical measure of how smooth the edge is. If the edge has *0-degree* continuity, then there are simply no holes, but there may be a crease. If the edge has *first-degree* continuity, then the first derivative of the surface is continuous across the edge. This derivative may be measured either *parametrically* or *geometrically*; the former is derived from the natural mathematical description of the patch, and the latter comes from the apparent (or visible) nature of the patches. Higher degrees of continuity refer to ever-smoother blends of the surface across the edge.

Convex A property of a shape in any dimension. A shape is *convex* if you can pick any two points in the shape, and connect them with a straight line that never goes out of the shape. In 3D, a sphere is convex, but a wineglass is not (pick a point in the bowl and a point on the base; the line between them is not entirely within the wineglass). A shape that is not convex is *concave*.

Convex Hull The smallest polyhedron that encloses a given object. The convex hull for a polygon can be created in 2D by drawing the polygon on a wooden board, and then driving in a nail at each vertex. Stretch a rubber band so it encloses all the nails, and then release it; the shape formed by the rubber band is the convex hull.

CSG See Constructive Solid Geometry.

Database amplification The notion that simple sets of parameters stored in a data structure or database may expand into quite complex objects by the application of suitable procedural processes. Examples are *fractal* and *graftal* shapes which require only a few numbers or syntactic rules to enable the generation of intricate and detailed models.

Data structure A computerized method of storing information such as a list, array, graph, network, etc.

Decomposition Breaking an object into smaller or simpler components, typically to facilitate the algorithmic processing of the original object by more general procedures.

Depth cueing The process of reducing the apparent brightness of an object the further away from the viewer it is positioned. This often enhances the perception of depth. The idea is that one is viewing the scene under foggy conditions, where the contrast diminishes with distance. If the background is black, this diminished contrast is equivalent to reduced brightness; if the background is gray, it is reduced saturation of the color.

- Difference** A Boolean operation over two objects where the resulting object is the set of points in the first object that are not contained in the second object. Difference is also called *subtraction*. This is commonly used to drill holes in a part by subtracting a cylinder from the part. See *symmetric difference*.
- Diffuse** Light which is reflected equally in all directions from a point on the surface of an object. The brightness of the surface does not, therefore, depend on the observer's position relative to the surface point.
- Digitization** The process of building a geometric model from existing drawings, physical objects, or models of objects. The three-dimensional coordinates are sensed, located, typed, or otherwise obtained from the source.
- Dilation** A scaling geometric transformation. Scaling of a point coordinate (multiplication by a constant) can occur independently in each dimension.
- Dithering** The process of trading spatial resolution for enhanced intensity or color resolution by distributing fewer shades over a region larger than a single pixel. The eye performs the spatial integration to produce the additional shades or intensities. Care must be taken to avoid producing distracting spatial patterns.
- Dynamics** Animation by the specification of forces and torques applied to masses, rather than their positions, velocities, and accelerations.
- Easing** A reduction in the acceleration or deceleration of a motion to present a smoother, more continuous movement. Also used to blend two motions together.
- Environment Map** A picture of the environment surrounding a model. It is usually used to approximate reflections off the model.
- Field of view** The solid angle across the viewing pyramid at the apex (center of projection) where the camera is located.
- Fractal** A geometric structure having the property that its frequency distribution is the same no matter what magnification (scaling) factor is applied to it. Examples typically cited are coastlines and clouds. In practice the term is also applied to any curve or surface obtained by the stochastic (random) perturbation of points on or in the surface as the shape is subdivided to (sub-)pixel size for rendering. Technically, a fractal is a mathematical construction that is infinite; any computer models based on these ideas are just approximations of true fractals.
- Geometric editor** An interactive program which permits the creation and modification of shapes by distinct operations on the geometric data structure representing an object, for example, instancing a primitive shape, transforming it, combining it with other shapes, etc.

Goniometric diagram This is a diagram that gives the intensity and color of the energy leaving a point light source for every direction in which light can leave the source. The diagram may be used with approximations for light sources with finite size.

Graftal Any curve, shape, or surface obtained by the deterministic or syntactic perturbation of points on or in some starting primitive as the shape is subdivided to (sub-)pixel size for rendering. Often graftals are generated by a *shape grammar*.

Hierarchy A tree in which nodes typically represent objects and edges represent relationships. For example, a human body could be represented by a hierarchy with the torso at the top (or *root*), and five children representing the arms, legs, and head. Each arm could be a smaller hierarchy of upper arm, lower arm, and hand; the hand is a hierarchy of five fingers, and so on.

Hither Clipping Plane See clipping planes.

Implicit surface A surface described by an equation. The surface contains all points which evaluate to a given number when plugged into the equation; usually this value is zero. Often the solutions must be found by numerical search techniques.

In-betweening See interpolation.

Interpolation The process of filling in information from a set of values. Interpolation can be used to generate plausible surface points for sparse data or for curves or surfaces defined with a small set of control points. It is also a way to generate positions for objects between **key frames** in an animation.

Intersection A Boolean operation over two objects where the resulting object is the set of points which lie in both objects simultaneously.

Key frames Cel animation images drawn by the animator which are interpolated algorithmically by the computer to generate inbetween images at the desired frame rate.

Key parameters Values of motion or other attribute parameters which are specified by the animator. Inbetween parameter values are typically generated by spline curves.

Kinematics Motion specification by position, velocity, and acceleration, rather than by forces. Usually applied to the manipulation of mechanisms involving articulated parts and various joints and constraints between the parts.

Level surface All solutions to the implicit equation $F(x) = k$ for some constant k and multidimensional vector x .

- Light source** The colloquial name for a luminaire.
- Luminaire** An object which emits light on its own, even if no other light is striking it from elsewhere.
- Local coordinate system** A reference frame positioned within or on an object to facilitate its definition, manipulation, and placement in a scene. The local coordinate system moves along with the object.
- Matte** A two-dimensional region (not necessarily connected) of an image which indicates where one image is to be composited with another.
- Mip-Map** A pre-processed form of a texture which can increase rendering speed. “Mip” stands for *multum in parvo*, Latin for “many things in a small place”. Similar to the *sum table*.
- Octree** A data structure for a spatial partition which divides space into eight equal cubes (*octants*). Each cube is recursively subdivided as needed to the level of detail of the model or the pixel size.
- Orthogonal** A synonym for *perpendicular*.
- Orthogonal projection** The view created by placing the camera at an infinite distance from the scene. The view direction creates a viewing parallelepiped whose cross-section is determined by the window. Objects of the same size at different depths from the camera appear the same size in the image.
- Particle system** A representation which consists of (large) sets of individual points under algorithmic or stochastic control. The points have color, a trajectory, and a history. They may be created, destroyed, or split. The history may be used in a single image to create a curvilinear shape, or in an animation to create a moving point.
- Patch** Almost any 3D curved surface may be thought of as a patch. Typically they are small bits of surface that are created by a mesh of *control points*, which influence the patch in the same way that they influence a spline. Patches are usually connected to preserve different types of continuity. See *control points, continuity*.
- Paths** The value of a parametric function over time. Typically used to describe the spatial position of an object, but also applicable to its orientation, or even attribute values.
- Perspective projection** The view created by placing the camera at a finite position (the center of projection) within the scene. The camera forms the apex of a viewing pyramid whose cross-section is determined by the view direction and the window or field of view. Objects of the same size

at different depths from the camera appear in different sizes in the image: the further objects being foreshortened and hence appearing smaller.

Polygon A closed and connected sequence of vertices and edges. Usually considered planar and simple (non-intersecting). Polygons may also have internal edge loops (creating holes) or multiple components.

Polyhedron A 3D solid whose boundary is made of polygons.

Potential function Typically an exponential function with a center point and a value which decreases as a function of radius from the center. A level surface of a potential function is a sphere of some radius. By analogy to electric fields, adjacent potential functions may be summed so that the level surface of a set of potential functions is a surface of constant “charge” and therefore represents a smoothed combination rather than hard-edge sphere intersections. See *algebraic surface* and *blob*.

Primitives The simplest unit of geometric object represented in a data structure. The geometric, measurement, and rendering algorithms operate over primitives such that the results can be combined to apply to more complex objects consisting of admissible combinations of primitives.

Procedural representation The geometric model is embedded into any convenient computational procedure, such as a formula, implicit equation, or arbitrary code. The procedure may generate data structures (e.g. a procedure to create a cylinder of given length and number of sides as polygons), provide data values (e.g. from the evaluation of a function describing a surface), or return parameter values (e.g. a wave model where the outputs are control points for curved surfaces representing portions of the wave). Very complex models such as mountains, and very regular models such as a bookshelf of books, are usually good candidates for a procedural model.

Quadratic surface A surface defined by algebraic equations with variables raised at most to the power 2.

Quadric surface See *Quadratic surface*.

Quadtree A partition of the plane into four quadrants, each of which may be recursively subdivided into quadrants to the desired level of object detail or pixel size.

Radiosity A shading process in which the visibility of any portion of a surface of an object is assessed relative to every other surface. Because all the illumination is pre-computed once, the database can be stored as colored primitives that represent their illumination, and real-time hardware may be used to simply display the primitives.

- Ray casting** The process of determining which object in a scene is first encountered by a ray emanating from the camera view location and passing through a pixel on the display screen. From the list of ray-object intersections the first visible intersection is chosen. The surface normal at this point is used to determine the shading value.
- Ray tracing** The process of determining the shade of a pixel in a scene consisting of arbitrary objects, various surface attributes, and complex lighting models. The process starts like ray-casting, but each ray is followed as it passes through translucent objects, is bounced by reflective objects, intersects objects on its way to each light source to create shadows, etc.
- Reflection** A scaling geometric transformation where one or more of the coordinate multipliers are negative. Also the effect of light bouncing off an object. See *reflectivity*.
- Reflectivity** A surface attribute of an object which causes incoming light to be reflected only at the angle opposite to the incidence angle about the surface normal.
- Rendering** The process of taking a geometric model, a lighting model, a camera view, and other image generation parameters and computing an image. The choice of rendering algorithm is dependent on the model representation and the degree of realism (interpretation of object and lighting attributes) desired.
- Rotation** A geometric transformation which causes points to be rotated about some axis through the origin by a given angle.
- Sampling** The process of selecting a finite number of places to probe or sample a continuous function or scene model in order to produce a dense enough set of values to produce a reasonable discrete approximation.
- Scan-line algorithm** A rendering technique which computes the image one scan line at a time taking advantage of the minimal changes from one scan-line to the next in an image. The algorithm reduces a three-dimensional visibility problem to a two-dimensional problem per scan-line.
- Scripts** Time dependent statements of events or action. The actions are motion parameters applied to objects. Often a script resembles a programming language but the timing of events may also be visualized in a graphical form.
- Shade tree** A custom shading procedure for a particular object. This procedure may use all sorts of information from the object and the environment to create complex surface descriptions, including simulated reflections and shadows. A shade tree may also be used to describe the output of a luminaire. See *luminaire* and *goniometric diagram*.

- Shading model** The algorithm used to determine the color of light leaving a surface given a description of the light incident upon it. The shading model usually incorporates the surface normal information, the surface reflectance attributes, any texture or bump mapping, the lighting model, and even some compositing information.
- Skew** A geometric transformation which causes points to be translated in one or more dimensions while remaining fixed in another. The amount of translation is a multiple of the value of the fixed coordinate.
- Slice** The process of dividing a model by a plane so that one portion may be removed to view the interior structure. It is also sometimes used to describe the contour of the object obtained in the cutting plane itself.
- Solid Texture** A texture which is defined at every point in space. An object textured this way appears to be “carved out” of a volume of material with this texture. Common examples include wood and marble.
- Spline** A curve, usually defined by a series of points called **control points**. In shipbuilding a spline is a long thin piece of wood suspended on two supports. At different places along the board weights are placed to create gentle curves; heavier weights exert more of an influence. Mathematical splines were originally developed to simulate this physical construction. Thus one speaks of the *weight* associated with a control point.
- Specular** The component of illumination seen at a surface point of an object which is produced by reflection about the surface normal. It depends on the observer position, whereas the diffuse component does not. Often this appears as a “highlight.”
- Stochastic sampling** A technique for anti-aliasing. The basic idea is to replace the regular sampling grid (such as one sample in the center of each pixel) with a more random, or stochastic, set of sample locations. The result is that regular aliasing artifacts such as jaggies are replaced by fuzzy, noisy edges. When enough samples are used the noise smooths out and none of the regular aliasing artifacts appear. The advantage of this technique is that it can eliminate the regular aliasing artifacts created by every regular grid.
- Stretch** A geometric transformation which deforms a object locally. It is usually applied to the control parameters of a curved surface.
- Sum Table** A pre-processed form of a texture which can increase rendering speed. Similar to the Mip-map.
- Super-sampling** The process of sampling a scene at a higher frequency than the pixel spacing. The resulting excess samples are averaged or filtered to

determine the actual pixel value. In contrast to **adaptive sampling**, super-sampling refers to taking many samples everywhere in the image, rather than just where they are needed.

Surface normal The vector which is perpendicular to the object surface and outward facing. Surface normals are one of the principal determiners of surface shading.

Surface tangent The plane which is tangent to the object surface. The tangent is primarily used when discussing the continuity of curved surface patches across patch boundaries.

Surface of Revolution A 3D shape created by a 2D curve that is rotated around an axis. Such shapes may be created on a lathe. Examples include candlesticks and wine glasses.

Swept Surface A 3D shape created by a 2D curve (called the *contour*) that is swept along some 3D curve (called the *path*). The contour may be scaled or otherwise transformed as it moves along the path. For example, a circular contour may be swept along an S-shaped path to make a snake.

Symmetric difference A Boolean operation over two objects where the resulting object is the set of points in the first object that are not contained in the second object and the set of points in the second object that are not contained in the first. See **difference**.

Texture A two- or three-dimensional pattern which is applied to the object surface during the shading computation. Textures may determine color, reflectivity, transparency, material, or any other factor that controls how a shape looks. A *displacement* texture can even mildly distort a surface. The advantage to texture is that the complexity of appearance does not have to be explicitly modeled in the object geometry.

Tessellation A partition of the plane or space into one or more non-overlapping primitives. The primitives are usually regular in size and shape. Common examples in 2D are squares and hexagons. In 3D, cubes are convenient.

Tile To cover a (possibly non-planar) surface with planar polygons. Also, a tile sometimes refers to a texture pattern over a square polygon.

Topology The mathematical notions that describe the underlying structure of an object without reference to its particular shape. Two meshes of rectangles of equal size have the same topology, regardless of the locations of their vertices. Topologists are fond of noting that there is no difference between a donut and a coffee cup; a flexible model of either one can be deformed into the other by stretching and pulling, without any ripping or cutting.

- Transmittance** A property of a surface describing the fraction of light that will pass completely through it. Thus a transmittance of 0 is opaque, 1 is invisible, and values inbetween offer various degrees of translucency.
- Translation** A geometric transformation which causes points to be moved by constant displacements in each coordinate.
- Union** A Boolean operation over two objects where the resulting object is the set of points which lie in either object.
- View direction** The direction in which the camera is pointing. It is the perpendicular to the viewing plane on which the image is formed.
- View up direction** The world direction which when viewed by the camera will appear to be vertical (up) in the image.
- Viewport** The sub-area of the screen or image formation device into which a graphical rendering or other drawing is placed.
- Visible surface** That part of an object which is visible from a given point of view.
- Visible line** The process of rendering or drawing only linear features (lines) to show at least only those edges or contour lines of surfaces which are in fact visible to the camera.
- Voxel** A three-dimensional volume element, usually a cube or box. It has a value (or density) and six sides and represents a finite sized point in a regular decomposition of space.
- Window** A rectangular subregion of the viewing plane which defines the top, bottom, and side clipping planes relative to the camera position.
- Wire-frame** The drawing of a model by tracing linear features such as edges or contour lines without attempting to remove invisible or hidden parts.
- World coordinates** The arbitrary coordinate reference frame in which object models are positioned to create a scene.
- World projection** A special background image built from the objects in the scene itself which is used to create convincing reflections on objects.
- Yon Clipping Plane** See clipping planes.
- Z-buffer** A rendering technique in which objects are scan-converted to pixel data and depth values and then inserted into a pixel array and a depth array. Where the object's pixel depth is less than that of the currently stored pixel/depth pair, the new data is written in, replacing the old. Advantages of the Z-buffer are that the objects need not be processed in

any special order, the image can be incrementally updated with additional objects at any time, and the scene can contain an unlimited number of objects.

Introduction to Rendering

Andrew Glassner
Microsoft Research

What is Rendering?

- Turning ideas into pictures
 - Communications tool
 - A means to an end
-
-
-

Rendering Roots

- The Visual Arts
 - Painting, Sculpture
 - Physics
 - Studying nature
 - Computers
 - Efficient algorithms
-
-
-

Approaches to Rendering

- Simulate Nature
 - Anything else
-
-
-

Simulating Nature

- Advantages
 - Precise goal
 - Path is clear
 - Practical value
 - Disadvantages
 - Ignores other styles
-
-
-

Photorealism

- The Big Winner so far
 - coined in 1968
 - just one school of art
 - alternatives emerging
-
-
-

////// *Today's view*

- Rendering as commodity
 - Buy, don't write
 - Still requires care
 - Efficiency
 - Accuracy
 - Effects
-
-

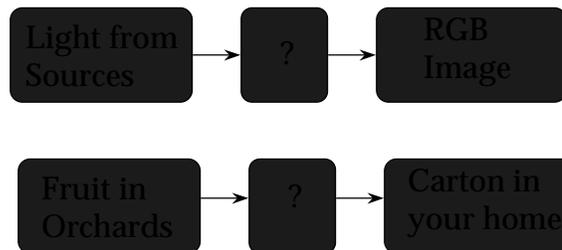
////// *Purpose of this talk*

- The basic ideas
 - Techniques
 - Tradeoffs
 - Vocabulary
 - High-level understanding
 - Not programming!
-
-

||||| The Big Picture

- Fake a photo (almost)
- Data into picture
 - Geometry
 - Surfaces
 - » Reflection
 - » Transparency
 - » Emission
 - Camera and film

||||| Juice Rendering



Movement, Loss, Combination, Change

What's in the carton?

- No peeking allowed!
- X-ray spy movie available
 - Everywhere on Earth
 - The last six months
 - Tedious and slow access

The Two Approaches

- Start at the orchard

- Work forward



- Start at the carton

- Work backward



Starting at the Orchard

- Follow the fruit
 - Pick a fruit and follow it
 - Picking, juicing, mixing, shipping
 - Identify every carton
 - Repeat for every fruit
 - Look up your carton
-
-
-

Starting at the Carton

- Follow the juice
 - Pick a molecule and follow it
 - Shipping, mixing, juicing, picking
 - Identify every fruit
 - Repeat for every molecule
 - Combine the right fruits
-
-
-

Which is better?

- Following the fruit
 - Long setup time without results
 - » Consumes time and memory
 - Can get any carton very quickly
- Following the juice
 - Can get this carton quickly
 - Other cartons basically start over

Your goal sets the strategy

- Many cartons to identify
 - Initial step very expensive
 - Carton-independent solution
 - Each carton cheap
- Only one carton to identify
 - Faster results
 - Carton-dependent solution
 - Each one is as expensive as the others

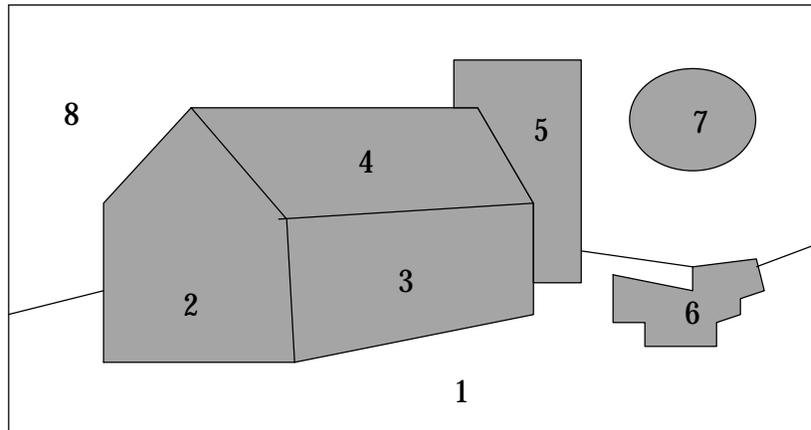
The Analogy

- A carton is a color dot
 - RGB = fruits
 - Big picture(s), long animation
 - Pre-processing pays off
 - Small picture(s), no animation
 - May be faster to be specific
-
-
-

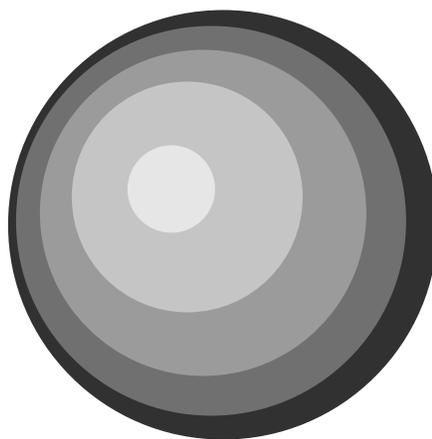
Shading and Visibility

- Visibility
 - What we see
 - Shading
 - How it looks
-
-
-

Visibility: Paint-by-numbers



Shading: Object colors



Local & Global Shading

- Use environment, or fake it?
- Local shading
 - Fake it, fast and dirty
 - Don't worry, be happy
- Global shading
 - Do it slow and correct
 - Worry, be happy

Why do it globally?

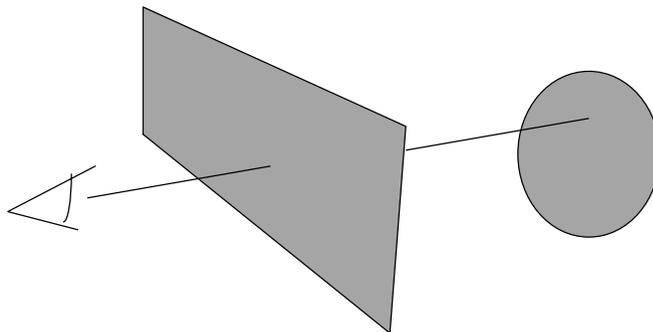
- Shadows
- Reflections
- Refraction (transparency)
- Emissions (light sources)
- Subtle touches

Why do it locally?

- Often close enough
- Fast!
- Hardware support
- Many effects can be faked
- Unlimited scene size

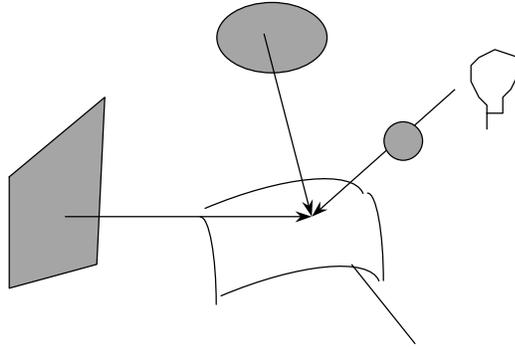
Ray Tracing 1

- Follow lines of sight



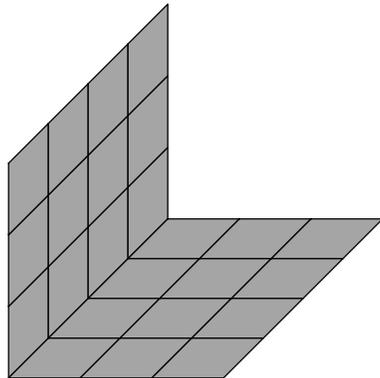
Ray Tracing 2

- Shoot new rays to find illumination



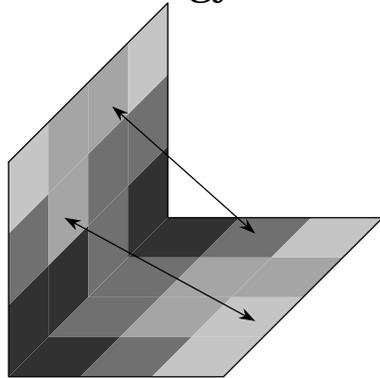
Radiosity 1

- Discretize the environment



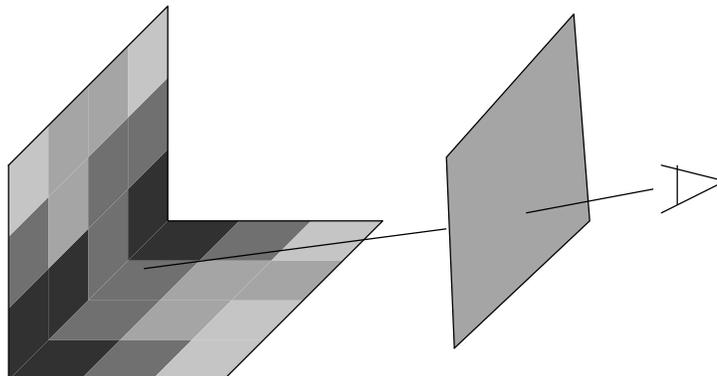
Radiosity 2

- Bounce energy back and forth



Radiosity 3

- View results



Hardware support

- Z-buffers
 - Infinite # of objects
 - Local shading
 - Fast
 - Many tricks available
-
-

Rendering at extremes

- Atoms
 - or sub-atomic particles
 - Galaxies
 - or the Milky Way
 - Relativity
 - Ocean depths
 - Inside people
-
-

Animation

- Film: 24 frames/second
 - 30 minutes = 43,200 frames
 - More for TV!
- Motion blur

Exposure

- Cameras
- Lenses
- Films
- Intended viewing conditions

Color 1

- The RGB Myth
 - The HSV Myth
 - The 8-bit Myth
 - The Constancy Myth
 - The Fifth Myth with pith
-
-
-

Color 2

- Devices vary
 - Printers vs. monitors
 - Perception matters
 - Metamers
 - Diet, recent conditions
 - Interpolation is tricky
 - RGB is not perceptually uniform
-
-
-

Images 1

- **Frame buffers**
 - Pixels
 - 8 bits/color
 - **Samples**
 - 1 sample/pixel
 - Supersampling
-
-
-

Images 2

- **Storage**
 - Components
 - » Integer, Floating point
 - Symbolic
 - **Compression**
 - Lossy (MPEG, JPEG)
 - Non-lossy (GIF)
 - Speed vs. space
-
-
-

Compositing

- Layering of images
 - Backgrounds, foregrounds
 - Real Projection
 - Mattes
 - Matte lines
 - Matte operators
 - Alpha buffers
-
-
-

The future 1

- Photorealism grows
 - More accuracy
 - More speed
 - Parallel algorithms
 - Subjective Rendering grows
 - New opportunities
 - More personal
-
-
-

The future 2

- Desktop animation
 - Sophisticated support
 - Standards and architectures
 - Rendering one piece among many

 - “You did *what*, Grandpa?”
-
-

Review

- Forwards or backwards
 - One frame or many
 - Shading
 - Visibility
 - Ray Tracing, Radiosity, Z buffers
 - Images
-
-

Hardware Basics

This document was taken from part of the *Hardware Basics* chapter of the introductory book *The Way Computer Graphics Works*, by Olin Lathrop, published by John Wiley and Sons, 1997, ISBN 0-471-13040-0.

Copyright Notice

This document is copyright 1997 by Olin Lathrop. No part of this document, including the images, may be re-published, re-transmitted, saved to disk, or otherwise copied except as part of the necessary and normal operation of the software used to view this document.

In this chapter, we'll talk about how to get all those neat pictures out of the computer so that you can see them. It's useful to understand some of this, since it affects what we might want to do, and how we go about it. There will also be some buzz-words that are worth being exposed to.

Display (CRT) Basics

This section is about the cathode ray tube (CRT), probably the most common means for making computer graphics visible. You may be used to calling this device a monitor. A monitor is really the whole unit that includes the outer shell, internal electronics, on/off switch, front panel twiddle knobs, etc. The CRT is the "screen" part where you see the picture.

What is a Cathode Ray Tube?

The CRT is one of the few types of vacuum tubes still in common use today. It works on the principle that some materials, called phosphors, emit light if you crash enough electrons into them. The face of a black and white CRT (we'll get to color shortly) is made of transparent glass coated on the inside with a continuous layer of this phosphor material. The rest of the CRT's job is to allow control over how many electrons hit the phosphor, and where they hit it.

A spot on the phosphor lights up brighter as more electrons hit it. It starts getting dimmer when electrons stop hitting it. It stops emitting light only a short time later.

The electron gun produces a thin stream of electrons. The electron flow rate, also called the beam current, is controlled by the monitor's electronics. The stronger the beam current, the brighter the phosphor will light up. The deflection yoke magnetically steers the electron stream so that it hits the phosphor at the desired spot. This is also under control of the monitor's electronics.

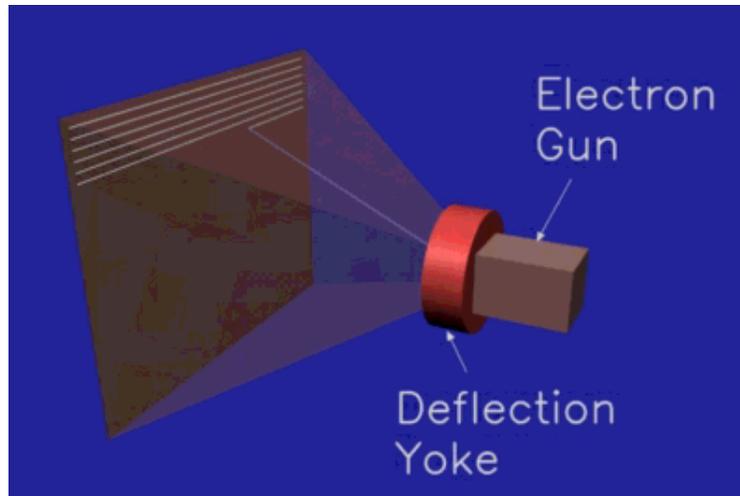


Figure 4 - Cathode Ray Tube Diagram

A thin stream of electrons is produced by the electron gun, which is aimed at a particular spot on the screen by the deflection yoke. The inside of the screen is coated with phosphors that light up when hit by the electron beam.

So, all a CRT really does is to cause a selectable spot on its screen to shine with a selectable brightness.

Raster Scan

But, when you look at a CRT monitor, you don't just see a few lit dots. You see whole areas lit. How is that done?

The dot that can be lit up is rapidly swept across the CRT face (by re-aiming the electron beam) in a pattern called a raster scan. The dot typically starts at the top left corner. It's then swept across to the top right corner. The electron beam is temporarily shut off while it's re-directed to the left side of the screen. The beam then scans across the screen just a little below the previous scan line. This process continues until the beam finally sweeps the bottom scan line, then the whole process is repeated.

An image is formed on the monitor screen by modulating the beam current as the lit dot is moved around. High beam currents make bright areas, and low beam currents make dim areas. The raster scan pattern maps nicely to the pixels in an image. Each horizontal traverse of the beam displays one horizontal row of pixels. The pixel values within each scan line are used to control the beam current as the beam sweeps across the screen.

Figure 4 shows six whole scan lines, with the seventh still being drawn. The point of a raster scan is to eventually hit every spot on the screen, so the scan lines overlap a little. Figure 4 shows spaces between the scan lines only to help you understand a raster scan.

So why don't we just see a dot flying around the screen instead of whole areas lit? There are two reasons for this. First, your eyes continue to perceive light for a short time after the light

has gone away. This is called persistence of vision. Second, a phosphor spot keeps glowing a little while after the beam goes away. This is called phosphor persistence. Therefore, as long as the spot is swept over the whole screen fast enough, it will appear as a steady area of light, instead of a flying spot of light.

So how fast is fast? Today's monitors typically scan the entire screen 60 to 80 times per second. If they went much slower, the image would appear to flicker.

Color

I can hear you thinking "I can almost believe this, but how is color possible? After all, electron beams don't come in different colors." No, they don't. What I've described so far is how black and white CRTs work. The basic process doesn't really support color. To get color, people have come up with some strange kludges and hacks. It's amazing to me color CRTs work at all.

While electron beams don't come in different colors, phosphors do. To make a color CRT, little phosphor dots for each of the three primary colors (red, green, blue) are arranged on the monitor screen. Then, three electron guns are used, one for each phosphor color.

The tricky part is to make sure each electron gun can only hit the phosphor dots for its color. This is done by arranging three dots of different colors in groups, called triads. Since there are now three electron beams coming from three separate guns, each hits the phosphors from a slightly different angle. A thin sheet called the shadow mask is suspended in front of the phosphors. The shadow mask has one hole for each triad, and is arranged so that each beam can only "see" the phosphor dots for its color. Take a look at Figure 5.

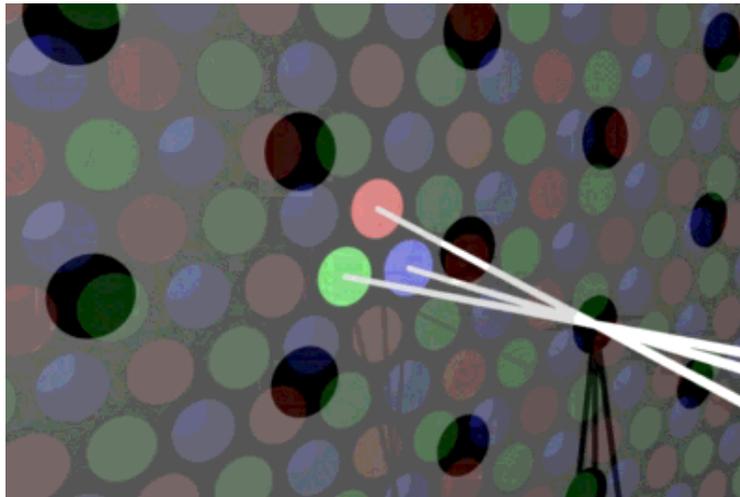


Figure 5 - CRT Shadow Mask and Phosphor Dots

The shadow mask is a thin sheet, shown here as semi-transparent, suspended in front of the phosphor dots. There is one hole in the shadow mask for each phosphor triad, even though it may not appear that way in this picture due to the perspective. Since the three electron

beams go thru the shadow mask holes from slightly different angles, each beam can only light up the dots for its color. The three white lines represent the electron beams passing thru a shadow mask hole to light up the center triad.

In case that just sounds too flaky to be true, see Figure 6.

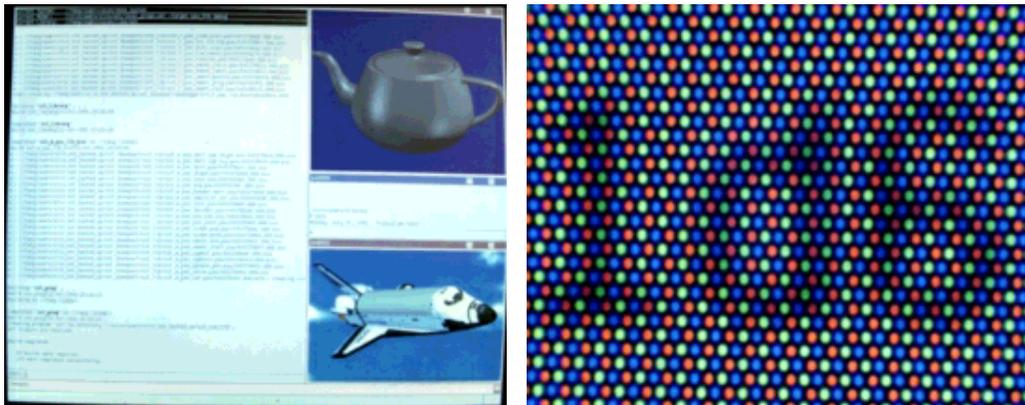


Figure 6 - CRT Photographs

These pictures are actual photographs of a color CRT face. The left picture shows the whole screen, whereas the right picture was taken from a small region near the center that was displaying the characters "ld6." Note how the image is really lots and lots of red, green, and blue dots. You see a continuous shade instead of dots, because individual dots are too small to see at the normal viewing distance. Your eyes blend the dots together, much like in the left picture. Try looking closely at a color CRT with a magnifying lens or a jeweler's loupe.

It's important not to confuse phosphor triads with pixels. Pixels are the individual color values that make up the image stored in the computer. Phosphor triads happen to be a hack to get CRTs to display colors. The whole mechanism of three beams, a shadow mask, and phosphor triads only exists to provide separate red, green, and blue color control over the lit spot on the CRT face. As long as there are enough triads so you can't see individual ones, you can think of the CRT face as being continuous. The triads are arranged in a hexagonal pattern, while pixels are in a rectangular pattern. A CRT could be made where the shadow mask and triads are rotated a bit from horizontal with little overall effect.

Does the whole mechanism of three beams, a shadow mask, and phosphor triads sound a bit flaky? It is. How do you make sure the three beams hit exactly the same spot as they are swept across the screen? What if they don't? What if the shadow mask is off a little and the beams don't just hit the spots for their colors? Well, these are real problems.

The degree to which all three beams line up to converge to one spot is called convergence. A monitor that is poorly converged looks blurry, and shows color fringes around the edges of objects.

The degree to which each beam only hits the phosphor dots for its color is called color purity. If a CRT has poor color purity, colors will look less vivid, and there may be patches of tint here and there. For example, the top right corner may be more reddish, whereas the top left corner more greenish.

All this is very sensitive to magnetic fields, since they can affect the electron beam paths. To prevent metal in the monitor chassis from becoming magnetic, any built up magnetism must be periodically removed. This process is called de-Gaussing, and is usually done automatically every time the monitor is turned on. Listen for a low hum lasting about a second right after a monitor is switched on.

Why Do We Care?

A basic understanding of what's going on inside a CRT can be a big help when you are buying color CRT monitors. In this section I'll briefly go over the CRT buzzwords you may see in catalogs or hear from salesmen.

Keep in mind that while many salesmen are quite knowledgeable, there are also far too many that don't really understand what they're talking about. Unfortunately, it's up to you to know what you want and to sort fact from fiction.

In general, any specification is measured to appear as favorable as possible, as long as there is some remote justification for doing so. It would be nice if specifications were for what end users really get. Yeah, right.

Size

Monitor sizes are measured in screen diagonal length. Computer monitor sizes range from about 12 to 21 inches. You might think that on a 12 inch monitor the upper right image corner would be 12 inches from the lower left image corner. Unfortunately, it's not that simple.

I'm writing this on a 17 inch monitor, but I measure only 16 1/4 inches between opposite inside corners of the bezel surrounding the screen. That's because 17 inches is the size of the bare CRT, not the final displayable area that I get to see. The monitor manufacturer buys 17 inch CRTs, and therefore claims to sell 17 inch monitors.

Worse yet, I can't even use all 16 1/4 inches of the visible CRT face. Most computer displays have a 5:4 aspect ratio, meaning they are 4/5th as tall as they are wide. Example 5:4 resolutions are 640x512, 1024x768, 1280x1024, and 1600x1280. After adjusting the image to the largest possible 5:4 area, I am left with only a 15 1/4 inch diagonal. I guess this means $15 \frac{1}{4} = 17$ in marketing math.

Dot Pitch

Dot pitch indicates how closely spaced the individual phosphor triads are, which relates to the smallest detail the CRT can possibly show, or resolve. (In practice, monitor resolution is dependent on many parameters.) Remember how there was one

shadow mask hole for every color triad (you can refer to Figure 5 for a reminder). The CRT can't resolve anything smaller than the distance between adjacent shadow mask holes. This distance is what's called the "dot pitch", even though it refers to triads, not dots.

The triads are arranged in a hexagonal pattern so that each triad has six neighbors. The distance to each of these neighbors is the dot pitch. See Figure 7 for a diagram of all this.

Typical dot pitch values are in the .2 to .3 millimeter range.

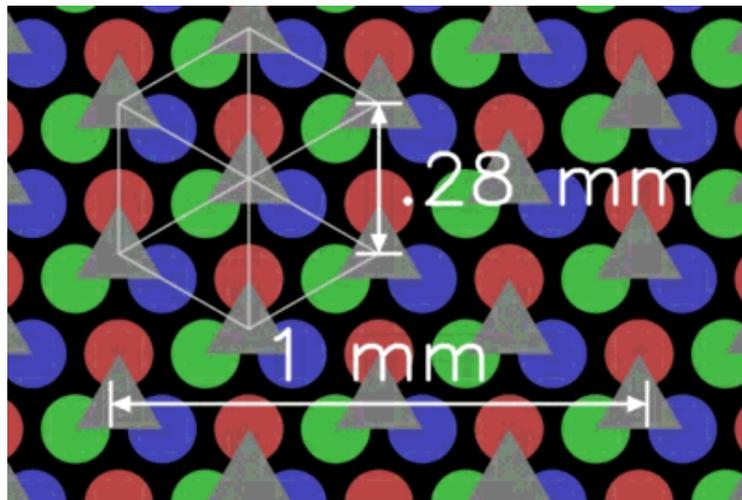


Figure 7 - Monitor Dot Pitch Measurement

This diagram shows how a monitor's dot pitch is measured. The monitor in this example has a dot pitch of .28 millimeters. The gray triangles mark triads so that you can see them more easily. The "cartwheel" in the upper left shows how each triad is the same distance from all six of its neighbors.

Triads Versus Pixels

There seems to be much confusion among computer users about the distinction between triads and pixels, and how they relate to each other. Remember that an image is a rectangular array of pixels. A quick look at Figure 7 should show you that pixels can't have a one-to-one relationship with triads, since triads are arranged in a hexagonal pattern.

Phosphor triads are just a means of making a CRT display colors. But, you can pretty much forget triads exist because they are deliberately so small as to present the illusion of a continuous color screen.

Pixels, on the other hand, are the digital values that are used to drive the monitor's analog controls. Each horizontal row of pixels in the display hardware

is used to modulate the electron beam currents for one horizontal sweep across the screen. The exact vertical placement of the beam sweeps, and the horizontal placement of individual pixels within each sweep is without any regard to the placement, spacing, and orientation of the phosphor triads. In other words, pixels just fall where they fall, and it works because the screen can be thought of as continuous, even though it happens to be made of lots of little dots. After all, most monitors have some diddle knobs that allow you to move the image up, down, left, and right, change its size, and sometimes even rotate it slightly. Clearly, pixels can be moved around without regard to the phosphor triads.

The spacing between phosphor triads, the "dot pitch", does affect the monitor's maximum possible resolution. In other words, the visual detail in a color CRT image stops increasing as more pixels are used, once the pixel spacing is about the same as the triad spacing. For example, let's consider a monitor with a dot pitch of .28 millimeters. This means the closest spacing between triads is .28 millimeters, which means there are no more than 36 triads per centimeter, or 91 triads per inch. If you've got a 1,280 x 1,024 display, that means your monitor needs to be at least 14 inches wide to truly resolve all the pixels. This comes out to a diagonal display area size of 18 inches, which would require at least a "19 inch" monitor to achieve. In practice, the advantage of denser pixels doesn't just suddenly stop when the dot pitch is reached, but it does start to seriously fall off. You will probably still perceive a better image at 1,280 x 1024 pixels on a 17 inch monitor with .28 dot pitch than at 1,024 x 800 pixels. But, it's unlikely this would be true on a 12 inch monitor with a .28 dot pitch.

Scan (or Refresh) Rate

Scan rate, or refresh rate, refers to how fast the electron beams are swept across the screen. There are really two scan rates, horizontal and vertical. The horizontal scan rate is the rate at which individual scan lines are drawn, and is of little interest to end users. Users are more concerned with how many scan lines there are and how often the whole screen is refreshed. Typical horizontal scan rate values are in the 15 to 100 Kilohertz range (15,000 to 100,000 times per second).

The vertical scan rate indicates how often the entire image is refreshed. This directly affects how much the image will appear to flicker. Most people will perceive a monitor image to be "flicker free" at vertical scan rates of 60 to 70 hertz (60 to 70 times per second) and higher.

You may be able to see a monitor flicker by looking at it out of the corner of your eyes. Humans are more sensitive to flicker at the periphery of vision than at the spot they are looking directly at. Try this with a regular (non-digital) television. These flicker at 60 hertz in North America and Japan, and 50 hertz most everywhere else. 50 hertz is so low that many people can see the flicker even when looking directly at the screen.

Interlacing

A monitor running in interlaced mode only refreshes every other scan line each vertical

pass. This means the entire image is refreshed every two vertical passes.

Why go thru all that trouble? The apparent flicker you see comes from the vertical refresh rate, whether all the scan lines or only half are displayed each pass. For the same apparent flicker, interlacing only requires half as many scan lines to be drawn. This reduces the data rate and relaxes requirements on the monitor and graphics hardware electronics, saving money. Each vertical pass during which half the scan lines are drawn is called a field. The field containing the top scan line is called the even field, and the other is called the odd field. Both fields together, meaning all the scan lines are drawn once, is called a frame.

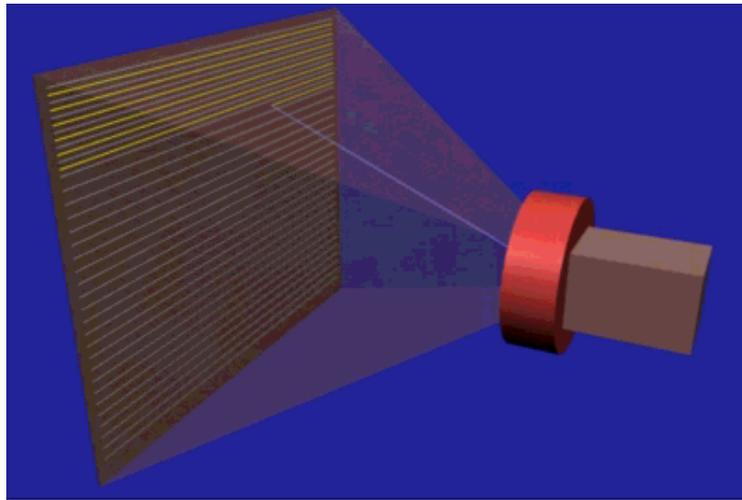


Figure 8 - Scan Line Interlacing

This diagram shows a frame being drawn part way thru the second, or odd, field. The first, or even, field scan lines are shown in gray, while the odd field scan lines are shown in yellow. Note how each scan line is drawn half way between two scan lines of the previous field.

So, briefly, interlacing is a means of reducing monitor and display hardware cost, while displaying the same image with the same apparent flicker. However, there's no free lunch. Thin horizontal lines in the image will flicker quite noticeably. Since they are on only one scan line, they only get refreshed at half the vertical refresh rate. Also, interlacing introduces one more thing that can go wrong. The image will be of poor quality if the beam sweeps of each vertical pass aren't exactly half way between each other.

Display Controller Basics

A display controller is a piece of computer hardware that takes drawing commands from the processor and drives the display. This is often called the video card or graphics card. See Figure 9. The output of a display controller are the video signals for the monitor.

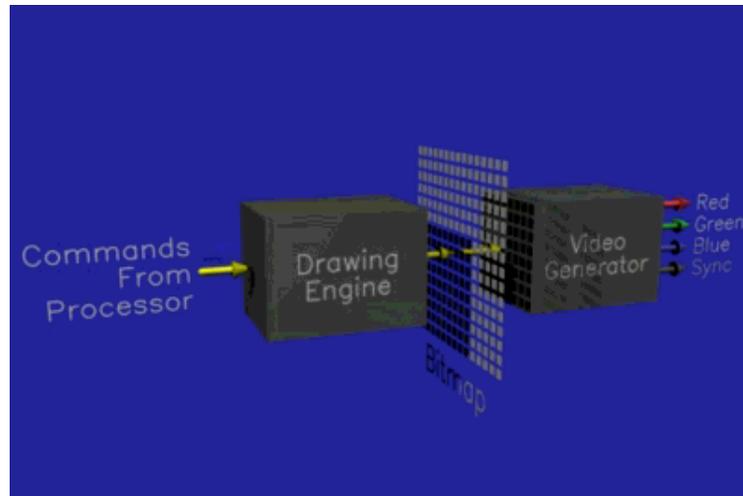


Figure 9 - Display Controller Block Diagram

The main display controller components are the drawing engine, bitmap, and video generator. These are described in the following sections.

The bitmap is the heart of any display controller. This is where the pixels are kept. The bitmap divides the remaining display controller into the drawing "front end", and the video "back end." We'll talk about those first, then get back to some more bitmap details.

Drawing "Front End"

The front end, or drawing engine, receives drawing commands from the processor. The front end figures out which pixels are being drawn, and what color, or value, they should be. The pixels are "drawn" by writing the new values into the bitmap.

The drawing command set can vary greatly from one display controller to another. To give you some idea, a typical command sequence for drawing a red rectangle from 15,10 to 34,24 might be: 1 - Set current fill color to red. 2 - Set current point to 15,10. 3 - Draw rectangle, width = 20, height = 15.

Most display controllers also allow the processor to directly read and write the pixels in the bitmap. The processor could have directly written red into all the pixels from 15,10 to 34,24 to write the same rectangle as before. However, the purpose of the drawing engine is to off-load this kind of work from the processor. Not only can the drawing engine do this task faster, the processor can go do something else once the drawing engine gets started on the rectangle.

Video "Back End"

The job of the video back end is to interpret the bitmap pixel values into their colors, and to create the video signals that drive the monitor so you can see the colors. The bitmap values are re-read each time the monitor image is refreshed. Since this typically happens 60-80 times per second, the bitmap is effectively displayed "live."

Color Lookup Tables (LUTs)

I mentioned before that one of the video back end's jobs is to interpret the bitmap pixel values into their resulting colors. This sounds a little silly. Why aren't the colors just stored in the bitmap directly? There are two reasons for this. The main reason is to require less bitmap memory. A secondary reason is to allow some correction for the weird things monitors can do to colors and brightness levels.

So, how does going thru an interpretation step save memory? Well, let's look at what it would take to store color values directly. As I mentioned before, it takes three numbers to describe a color. The standards for video signals that drive computer monitors use the RGB color space, so the three numbers would need to be the red, green, and blue color components. In computer graphics, we think of RGB color components as being "continuous" (you can't distinguish individual levels anymore) when there are at least 256 levels per RGB component. Since 256 levels requires 8 bits ($2^8 = 256$), or one byte, a full color requires three bytes. If your bitmap has a resolution of 1024x800 pixels, that would require about 2.5 megabytes for the bitmap. Memory usually comes in standard sizes, so you'd probably end up with four megabytes in your bitmap. (No, this isn't stupidity. There are good reasons for this, but they're beyond the scope of this book).

The cost of low end graphics boards is usually dominated by the cost of the bitmap memory, so we'd like to reduce the amount of this memory. Three bytes per pixel lets us store any color in any pixel, but do we really need this? Unless you are doing imaging, the answer is usually "no." Look at a typical screen with a few windows, text, menus, etc. How many different colors do you see? Probably not more than 16. Suppose we numbered each of these colors from 0 to 15. We would then need only four bits per pixel in the bitmap, but we'd have to interpret the color numbers into their real colors to generate the final RGB video signals.

In practice, we usually use eight bits per pixel instead of the four in the example. Eight bits allows up to 256 different colors on the screen at the same time. That's more than enough for the basic user interface, but also allows some way to see images, supports games, etc. 256 simultaneous colors requires one byte per pixel. The entire 1024x800 bitmap would then fit into just one megabyte with room to spare. Note that we've reduced the bitmap memory from four to one megabyte at a price. First we can only display 256 colors simultaneously, and second, we now have to interpret the color numbers into real RGB colors.

The interpretation job is done in the color lookup table, often just called the LUT. The LUT converts the color numbers, usually called the color index values or pseudo colors, from the bitmap into their assigned RGB colors. In our example, the LUT has 256 entries, since that's how many possible color index values there are. Each entry holds a 24 bit (8 bit per color component) RGB value.

True Color, Pseudo Color

A system that stores RGB values directly in the bitmap is called a true color system,

and one that stores color index values is called a pseudo color system. Figure 10 and Figure 11 show how the final displayed color value is determined for each pixel.

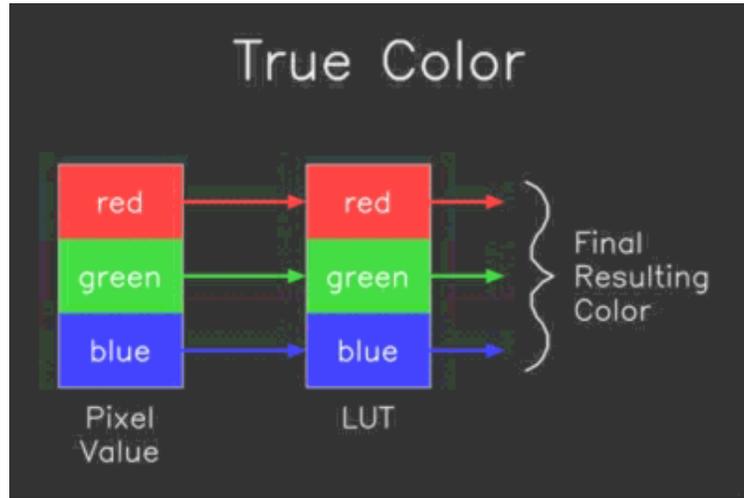


Figure 10 - True Color Interpretation

True color is the conceptually simple color configuration. The actual, or "true", pixel color is stored directly in each pixel. The color lookup table, or LUT, is not necessary in a true color system. It is usually present because most true color systems also support pseudo color where the LUT is needed. In true color mode, the LUT is usually loaded with values so that it has no net effect. It is sometimes used to compensate for artifacts introduced by monitors, and for special effects.

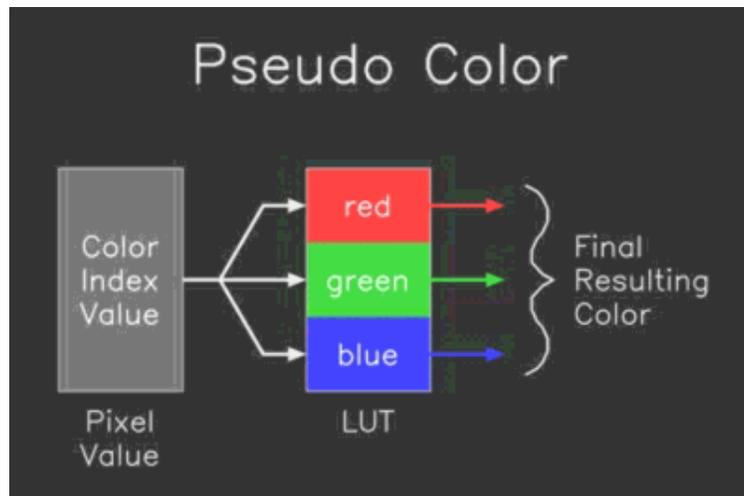


Figure 11 - Pseudo Color Interpretation

In a pseudo color configuration, each pixel holds an index into the color lookup table instead of a true color value. The color lookup table is required, and converts the color

index values into the true RGB color values.

While a lookup table (LUT) is required in a pseudo color system, many true color systems also use them. In that case, they can be used to compensate for some artifacts introduced by the monitor, or for special effects. In practice, most true color lookup tables are just loaded with "straight thru" data, and you can usually forget them.

Let's do some examples to make sure the true color versus pseudo color distinction makes sense.

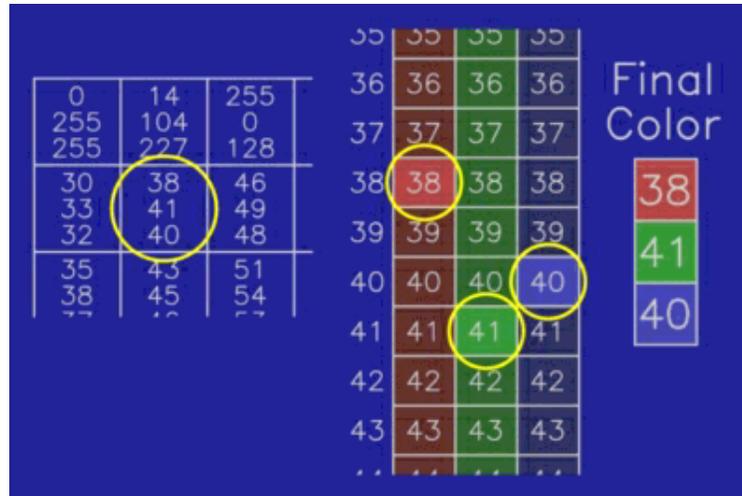


Figure 12 - True Color Interpretation Example

In this example we are trying to determine what the final visible color is for the circled bitmap pixel on the left.

Since this is a true color example, each pixel contains separate red, green, and blue values. In this case there are eight bits per color component per pixel, so color components range from 0 to 255. The selected pixel contains red 38, green 41, and blue 40. The corresponding

LUT entries are found (the left column on the light blue background) separately for each of the color components. The resulting final color values from the LUT are circled and shown on the right.

Note that in this example the LUT values are such that the final color is the same as the pixel values. This is usually the case in true color because there's usually no need for an additional interpretation step between the pixel values and the final color values.

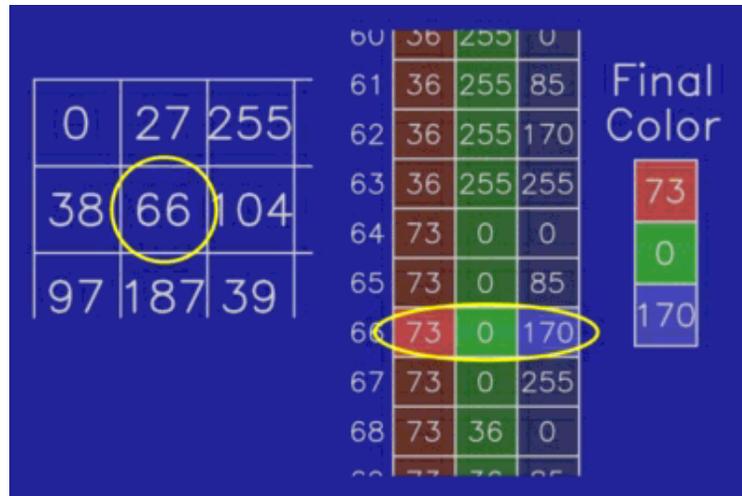


Figure 13 - Pseudo Color Interpretation Example
 In this example, a pseudo color is converted to a final color value. The pseudo color value from the selected pixel is 66. Therefore, the final color value is taken from LUT entry 66 for all color components, as shown.

Bitmap

The bitmap is the two-dimensional array of pixels that the drawing front end writes into, and the video back end reads from. Because frequent and high speed access is required to the bitmap, it is always (except for some rare specialty systems) implemented as a separate memory right in the display controller. You have no reason to care how the bitmap is implemented, only the price and performance of the overall display controller.

DRAM Versus VRAM

Although you shouldn't have to care, you can sometimes choose between DRAM and VRAM. What are those, and what do they do for you?

DRAM stands for Dynamic Random Access Memory. I won't go into what that means, except that DRAM is the "normal" kind of memory that is also used to make the main memory in your computer. VRAM stands for Video Random Access Memory, and is specifically designed to function as bitmap memory.

The drawing front end can independently write into a VRAM bitmap while the video back end is reading the pixels. In a DRAM bitmap, the front and back ends have to share. In some DRAM configurations, the back end can hog the bitmap up to 80% of the time. This doesn't leave much for the front end, and slows down drawing operations. Of course there's always a tradeoff, which in this case is price. VRAMs cost about twice what DRAMs cost for the same amount of memory.

What do I recommend getting? That's kinda like asking whether I recommend a station wagon or a sports car. In general, though, I wouldn't shell out the extra moola for

VRAM unless I knew I'd be running drawing-limited applications where performance is important. If you're not sure about that, just get DRAM.

What's in Hardware and What's in Software

Take another look at Figure 9. Note that the drawing engine isn't absolutely necessary, as long as the processor has direct access to the bitmap. Such a system wouldn't need to lack in features. It would be low cost but slow. At the other extreme, a system might have full hardware support for everything from simple lines to fancy 3D operations and drawing commands. This would be faster but more expensive.

In practice, even low end systems usually have hardware support for simple 2D drawing. The incremental cost of adding such a drawing engine is small compared to the bitmap and the video back end cost. Such a system is sometimes referred to as a 2D display controller or graphics board, or GUI engine. GUI stands for "graphical user interface" and refers to these kinds of operations.

There are systems available with just about any imaginable tradeoff between what's in hardware and what the software must do. Marketing types, however, like fancy labels to make their product sound more sophisticated than the next one. Some "standard" names have emerged for some configurations. I'll make you aware of them, but keep in mind this is a moving target since companies can (and often do) make up new names, and use old names in new ways.

I've already mentioned 2D or GUI engine. This usually means a minimal drawing engine that's good at simple 2D lines, points, rectangles, pixel copies, and maybe some polygons (we'll get into what these are in the next chapter). That's all that's needed by most window systems for menus, text, popups, etc.

A 2 1/2 D display controller is intended for drawing 3D objects, but doesn't have true 3D capability. It provides the 2D support needed for 3D drawing. This usually includes allowing the color to vary across the object being drawn, dithering, and Z buffering.

A full 3D display controller understands true 3D commands. It must do transformations, lighting, and other advanced effects that don't make sense to talk about until you've read the Rendering chapter.

Technology keeps marching on. In the current trend, the cost of logic for implementing drawing engines is falling faster than the cost of the bitmap and the video back end. If this continues, we will see ever more capable "low end" systems. Who knows what tomorrow brings?



An Introduction to Computer Animation

Andrew Glassner
Microsoft Research

88 1 199



Why Animation Works

- **Many still images in rapid succession**
- **Persistence of vision**
 - **Must overcome flicker**

////// Animation is Expensive!

- **30 frames/second**
 - **30 minutes = 54,000 frames**
 - **5 minutes/frame, 12 hours/day ~ 1 year**
 - **Limited animation**
 - **Computer-assisted animation**
-

////// Thinking About Animation

- **Low level: Individual frames**
 - **Medium level: Sequences and scenes**
 - **High level: Story and Message**
 - **Computer helps all three levels**
-

Traditional 2D Animation

- **Hand-drawn *cels***
 - **Stacks of cels over background**
 - **Only redraw cels that change**
 - Limited animation
 - **Experimental forms**
-

Traditional 3D Animation

- **Shoot individual frames with camera**
 - **Stop motion**
 - King Kong, Wizard of Space and Time
 - **Puppetry**
 - Claymation
 - **Experimental forms**
-

Computer-Assisted Animation

- **2D**
 - Animator draws frames
 - Computer helps ink & paint
 - **3D**
 - Animator builds models, sets poses
 - Computer interpolates and renders
 - **Computer helps with compositing**
-

2D Computer Animation

- **What gets interpolated?**
 - Strokes
 - Outlines
 - Colors
 - **3D relationships hard to judge**
 - **Timing based on exposure sheets**
 - **High-quality compositing**
-

2D Morphing

- **Image interpolation**
 - **Feature matching**
 - **Multi-layer**
-

3D Computer Animation

- **What gets interpolated?**
 - **Shape geometry**
 - **Shape appearance**
 - **Light source information**
 - **Cameras**
 - ***Anything!***
 - **Model transformed then rendered**
-

Object Animation

- **Location**
 - **Geometry**
 - **Transformations and deformations**
 - **Appearance and textures**
 - **Light sources**
-

Camera Animation

- **FOV**
 - **Focal length**
 - **Position & orientation**
-

Object Interpolation

- **Parameterized transformation**
 - Rotation
 - Scaling
 - Translation
 - **Interpolate parameters**
 - **Build new transformations**
-

3D Animation Methods I

- **Low-level manual**
 - **High-level manual**
 - **Automated**
-

Parametric Interpolation

- Any number of parameters
 - User-specified smoothness
 - Transformations & deformations
 - Parameter source
 - User
 - Measured data (rotoscoping)
 - Procedural
-

Photograph

**A simple arm animated
parametrically**

Kinematics

- **Specify position and time**
 - **Give velocity, acceleration of parts**
 - **Nested interpolation**
 - **Each transformation accessible**
-

Photograph

**A simple arm animated
kinematically**

Inverse Kinematics

- **Goal-driven**
 - **Supports *constraints***
 - **Control what you care about**
 - Let computer fill in the rest
-

Photograph

A simple arm animated with inverse kinematics

Constraints

- **Parametric restrictions**
 - **Aids in inverse kinematics**
 - Put the foot on the ground
 - **Restricts user to “sensible” poses and motion**
 - **Can be frustrating**
-

Dynamics

- **Physical simulation of motion based on physics**
 - **Requires information for each object**
 - Mass, center of mass, moments of inertia, friction, etc.
 - **Very accurate motion**
 - **Difficult to control**
 - Results often surprising
-

Simulation

- **Precompute model parameters**
 - **May be expensive**
 - **Many simulation methods available**
 - **Complex motion such as flocking**
-

Scripting

- **Programming language for motion**
 - **Result of simulation with constraints**
 - **Can include 2D transitions between scenes**
-

Artificial Intelligence

- **High-level “director’s language”**
 - **Converts words to motion**
 - **Supports planning and constraints**
 - **Can use expert knowledge**
 - **Computer storytelling?**
-

Combined Methods

- **Key-framing major components**
 - **Simulation for details**
-



Conclusions

- **The computer is not the animator!**
 - **Many types of motion control**
 - **Different skill sets required**
 - **Good results come from good people**
 - **Animation is an art**
-

Computer Animation Techniques

Norman I. Badler *

1 Introduction to Animation

Computer animation in the most general sense is the computational control of images or objects over time. The objects may be two-, three-, or even multi-dimensional while their form may be represented in numerous different ways in the computer program. Images are usually two-dimensional, but may have color, depth, and compositing information associated with each pixel. A sequence of images comprises an animation, whether recorded on film or video, or simply viewed interactively. The control of an animation is based on the manipulation of parameters of the image or object. Parameters may describe image background, compositing operations, and layout; or object shape, color, texture, position, orientation, motion paths, and so on. As the parameters change over time, the corresponding attributes of the image or object change to produce the animation.

The principal task of the animator is to select and control the "proper" parameters to produce the desired effect. Certain applications may dictate a more image-based control approach, while others may require more explicit object models. We will focus on the latter topic, as it offers the more richly variable domain of the two. This restriction is both an advantage and a disadvantage for the animator: on the one hand the explicit nature of the object models and the controllable parameters offers the possibility of complete control over a purely synthetic, often three-dimensional world; on the other hand the more classical (manual) techniques of art and animation relate to the image surface and do not necessarily carry over to the model domain. Thus, for example, creating an animation of a fly-through a proposed building is easy with a three-dimensional model, but trying to animate the initial visualization stages of its design from hand-drawn sketches is much more difficult. Whereas the skilled artist has no difficulty working with pencil and paper, the requirements of object models and explicit three-dimensional formulations is an acquired skill with technological implications.

*Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA 19104 USA

Determining the best, or at least workable, methods of defining changing values for the large numbers of parameters potentially involved in an animation is no simple task. Once the parameters have been decided upon, the animation itself is produced by evaluating the object appearance and position relative to the parameter values as they change over time. Unfortunately, there are really no universally agreed upon parameter sets; rather, the definition of parameters for manipulating object models for animation is a rapidly evolving field. One of our tasks here is to provide some idea of the scope and breadth of such animation control techniques.

2 Animation Techniques

Animation techniques presently in use include image interpolation, parametric interpolation, kinematics, inverse kinematics, constraints, dynamics, simulation, scripting systems, and Artificial Intelligence control.

3 Image interpolation and Morphing

In image interpolation the two-dimensional drawing of an object's boundary and features is transformed over time by moving (interpolating) points of the drawing between specified positions of those points in a sequence of keyframes. Areas are manipulated by moving their boundary shapes [42]. The interpolation method is usually linear, though modifications of the interpolation rate [29] and actual curved paths [36] have been used. The technique is intended to model the two-dimensional image manipulations performed by conventional animators. It is difficult to judge three-dimensional relationships, and motion control is implicitly embedded into image sequence changes by the keyframes.

4 Parametric interpolation

In parametric interpolation, desired attributes of the object model are parameterized so that they may be altered over time. As noted above, parameters may include object geometry, topology, color, surface attributes, textures, position, and orientation. Light sources and their attributes, camera shots, and other image features may also be parametrically specified and changed. The parameters are given key values at various time points and linear or smoothly varying curves are fit to the data points and interpolated to determine inbetween values [41, 24, 43]. Motion of an object along a given path is a good example of parametric interpolation.

Parameter values may be determined by direct measurement of some real motion, by algorithm, or by user determined key values. The number of parameters required for a character or figure animation can easily number in the

hundreds. The interaction between the parameters may become unmanageable unless considerable care is taken to either make the parameters independent [33] or create "macros" of pre-determined parameter sets [9].

5 Kinematics

Rather than specify key positions and interpolate, the animator might specify a starting position and a function of time which describes the changes to the parameter [28]. For example, the position of a ball in a trajectory can be parametrically controlled by evaluating its equation of motion over time. There is no explicit key "position" involved; the ball will continue to follow the trajectory "forever." Such control methods are kinematic in the sense that the motion of the object is controlled by functions of position, velocity, or acceleration.

By allowing motion information to be described relative to some other, perhaps moving, coordinate reference frame one can describe the motion of a part with respect to a parent object. For example, car wheels rotate with respect to the car axles which are attached to the car frame. By traversing the resulting tree structure of objects and sub-objects, the leaf nodes are positioned by composing the transformations nested along the tree edges. "Local motor programs" can be created to produce complex figure motion such as walking or jumping by parametrically controlling the timing of joint angle (position) transformations [50].

6 Inverse kinematics

Recently inverse kinematics has become increasingly important for human figure and animal behaviors. Inverse kinematics refers to the positioning of a jointed structure, such as an arm, torso, or leg, by defining the Cartesian space goal (3D position and 3D orientation) for an end effector and using an algorithm to determine reasonable positions and orientations for the intermediate joints of the linkage. While used in robotics for some time, inverse kinematics has only recently been subsumed into figure animation system as a powerful tool for decreasing the number of parameters requiring control by the animator [4, 26, 25, 17, 15, 54, 34, 35]. Besides reaching actions, inverse kinematics can be used to make the foot follow a specified spatial path during gait (walking) [15]. A principal difficulty is determining a "reasonable" set of joint angles when the chain is redundant: that is, more than six degrees of freedom exist in the chain. Different solutions are possible, e.g. utilizing the "null space" of a numerical solution to minimize joint angle change [17], considering global optimization criteria [48, 16], or maximizing comfort based on joint strength [27].

7 Constraints

The use of end effector goals and inverse kinematics leads naturally to a desire to specify other types of goals for objects and articulated figure parts. Since specifying several goals may easily lead to overconstrained systems (more constraints than degrees of freedom), a solution is usually obtained by some "non-procedural" minimization or iterative process. The constraints may take several forms, including relationships, boundary conditions, formulas, potential functions, or spring forces. They aid kinematic specifications and offer global control subject to local variation [48, 7]. Constraints have been applied historically [44] as well as more recently in geometric design [31, 40] and articulated figure positioning [32, 3]. Very general energy minimization approaches to object manipulation and animation are yielding very interesting results with such things as "self-assembling" towers and realistic chains [49, 6].

8 Dynamics

A dynamics approach to animation uses forces and moments to drive the motion of objects. The benefit is in producing motions that are physically reasonable and correct; the limitation is that all the forces must be determined. It is not obvious whether it is easier to work with forces rather than kinematics and constraints, though experiments show that animation with dynamics is able to assist in modeling force-based concepts such as balance, support, free-swinging movement, and object-to-object interaction [46, 1, 30, 20]. Any animation using dynamics must essentially rely on a frame-by-frame simulation to compute the incremental positions of the involved objects [19]. It appears that dynamics control is difficult unless combined with kinematics and constraints [46, 2, 22, 6, 47].

9 Simulation

Given a description of a process involving object interactions or parameter relationships that cannot be pre-computed, a simulation approach is needed. Here a model of some process is built in a suitable representation, then executed incrementally. Changes in the state of the process cause variations in the course of the simulation. The dynamic nature of the simulation process allows very general situations to be modeled and animated. Simulations usually offer discrete or continuous models; the latter are comparable to the requirements for dynamics animation. Other formalisms such as Petri nets and data flow diagrams may be used and compiled into discrete simulation models. There is a special interest in real-time simulated behaviors for systems and models [53, 52].

10 Scripting systems

A scripting system is essentially a programming system or language in which arbitrary changes to program variables can be invoked. The time dimension may be provided explicitly by the order of execution of the program, but more frequently the parameter changes are given times or temporal relationships and then posted in an event list for simulation-style execution [28]. Alternatively, the object behaviors may be encapsulated in an object-oriented programming paradigm [23, 37]. Scripts may be either language [18, 21] or graphics-based [12, 14] and offer an animation interface apparently rather like a series of director commands.

11 Artificial Intelligence control

Since there is no clear advantage in all situations to any of the above methods, there may be a need to organize several techniques to accomplish particular animation tasks. For example, human animation may require kinetics, dynamics, constraints, and a process model for the task being performed. While such systems are still evolving, several advantages are already apparent: general world model information, object interactions, rule-based action description, motion planning, and simulatable models [11, 51, 13, 39, 45, 10, 38, 52, 8, 5]. This approach may provide the only effective way to deal with the vast numbers of parameters involved to complex animations.

References

- [1] William Armstrong, Mark Green, and R. Lake. Near-real-time control of human figure models. *IEEE Computer Graphics and Applications*, 7(6):52–61, June 1987.
- [2] Norman I. Badler. A representation for natural human movement. In J. Gray, editor, *Dance Technology I*, pages 23–44. AAHPERD Publications, Reston, VA, 1989.
- [3] Norman I. Badler, Kamran Manoochehri, and G. Walters. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics and Applications*, 7(6):28–38, 1987.
- [4] Norman I. Badler, Joseph O’Rourke, and Bruce Kaufman. Special problems in human movement simulation. *Computer Graphics*, 14(3):189–197, July 1980.
- [5] Norman I. Badler, Bonnie L. Webber, Jugal K. Kalita, and Jeffrey Esakov. Animation from instructions. In Norman I. Badler, Brian A. Barsky, and

- David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 51–93. Morgan-Kaufmann, San Mateo, CA, 1991.
- [6] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22(4):179–188, 1988.
 - [7] Lynne S. Brotman and Arun N. Netravali. Motion interpolation by optimal control. *Computer Graphics*, 22(4):309–315, 1988.
 - [8] Tom Calvert. Composition of realistic animation sequences for multiple human figures. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 35–50. Morgan-Kaufmann, San Mateo, CA, 1991.
 - [9] Tom Calvert, J. Chapman, and A. Patla. Aspects of the kinematic simulation of human movement. *IEEE Computer Graphics and Applications*, 2(9):41–50, Nov. 1982.
 - [10] Jeffery Esakov, Norman I. Badler, and M. Jung. An investigation of language input and performance timing for task animation. In *Graphics Interface '89*, pages 86–93, San Mateo, CA, June 1989. Morgan-Kaufmann.
 - [11] Steven Feiner. APEX: An experiment in the automated creation of pictorial explanations. *IEEE Computer Graphics and Applications*, 5(11):29–37, Nov. 1985.
 - [12] Steven Feiner, David Salesin, and Thomas Banchoff. Dial: A diagrammatic animation language. *IEEE Computer Graphics and Applications*, 2(7):43–54, Sept. 1982.
 - [13] Paul A. Fishwick. *Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction*. PhD thesis, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1986.
 - [14] D. Fortin, J.-F. Lamy, and D. Thalmann. A multiple track animator system for motion synchronization. In N. I. Badler and J. K. Tsotsos, editors, *Motion: Representation and Perception*, pages 311–317. Elsevier, North Holland, New York, NY, 1986.
 - [15] Michael Girard. Interactive design of 3D computer-animated legged animal motion. *IEEE Computer Graphics and Applications*, 7(6):39–51, 1987.
 - [16] Michael Girard. Constrained optimization of articulated animal movement in computer animation. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 209–232. Morgan-Kaufmann, San Mateo, CA, 1991.

- [17] Michael Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures. *Computer Graphics*, 19(3):263–270, 1985.
- [18] Julian E. Gomez. Twixt: A 3D animation system. In *Proc. Eurographics '84*, pages 121–133, New York, NY, July 1984. Elsevier Science Publishers B.V.
- [19] Mark Green. Using dynamics in computer animation: Control and solution issues. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 281–314. Morgan-Kaufmann, San Mateo, CA, 1991.
- [20] James K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, August 1988.
- [21] Pat Hanrahan and David Sturman. Interactive animation of parametric models. *The Visual Computer*, 1(6), 1985.
- [22] Paul M. Isaacs and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints. *Computer Graphics*, 21(4):215–224, 1987.
- [23] Kenneth M. Kahn and Carl Hewitt. Dynamic graphics using quasi-parallelism. *Computer Graphics*, 12(3):357–362, 1978.
- [24] Doris H. U. Kochanek and Richard H. Bartels. Interpolating splines with local tension, continuity, and bias control. *Computer Graphics*, 18(3):33–41, 1984.
- [25] James U. Korein. *A Geometric Investigation of Reach*. MIT Press, Cambridge, MA, 1985.
- [26] James U. Korein and Norman I. Badler. Techniques for goal directed motion. *IEEE Computer Graphics and Applications*, 2(9):71–81, Nov. 1982.
- [27] Philip Lee, Susanna Wei, Jianmin Zhao, and Norman I. Badler. Strength guided motion. *Computer Graphics*, 24(4):253–262, 1990.
- [28] Nadia Magnenat-Thalmann and Daniel Thalmann. *Computer Animation: Theory and Practice*. Springer-Verlag, New York, NY, 1985.
- [29] L. Mezei and A. Zivian. ARTA, an interactive animation system. In *Proc. IFIP Congress*, pages 429–434. North Holland, 1971.
- [30] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, 1988.
- [31] Greg Nelson. Juno, a constraint-based graphics system. *Computer Graphics*, 19(3):235–243, 1985.

- [32] Joseph O'Rourke and Norman I. Badler. Model-based image analysis of human motion using constraint propagation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2(6):522–536, Nov. 1980.
- [33] Frederic Parke. Parameterized models for facial animation. *IEEE Computer Graphics and Applications*, 2(9):61–68, Nov. 1982.
- [34] Cary Phillips, Jianmin Zhao, and Norman I. Badler. Interactive real-time articulated figure manipulation using multiple kinematic constraints. *Computer Graphics*, 24(2):245–250, 1990.
- [35] Cary B. Phillips and Norman I. Badler. Interactive behaviors for bipedal articulated figures. *Computer Graphics*, 25(4):359–362, 1991.
- [36] William T. Reeves. Inbetweening for computer animation utilizing moving point constraints. *Computer Graphics*, 15(3):263–269, Aug. 1981.
- [37] Craig W. Reynolds. Computer animation with scripts and actors. *Computer Graphics*, 16(3):289–296, July 1982.
- [38] G. Ridsdale and T. Calvert. Animating microworlds from scripts and relational constraints. In *Proc. of Computer Animation 90*, 1990.
- [39] Gary Ridsdale, S. Hewitt, and Tom W. Calvert. The interactive specification of human animation. In *Proc. Graphics Interface '86*, pages 121–130, Vancouver, Canada, 1986.
- [40] J. R. Rossignac. Constraints in constructive solid geometry. In *Proceedings of Workshop on Interactive 3-D Graphics*, pages 93–110. ACM, 1986.
- [41] Kim L. Shelley and Donald P. Greenberg. Path specification and path coherence. *Computer Graphics*, 16(3):157–166, July 1982.
- [42] Peter Sorenson. Morphing magic. *Computer Graphics World*, 15(1):37–42, Jan. 1992.
- [43] Scott Steketee and Norman I. Badler. Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control. *Computer Graphics*, 19(3):255–262, 1985.
- [44] Ivan E. Sutherland. SKETCHPAD: A man-machine graphical communication system. In *SJCC*, page 329, Baltimore, MD, 1963. Spartan Books.
- [45] Jane Wilhelms. Toward automatic motion control. *IEEE Computer Graphics and Applications*, 7(4):11–22, April 1987.
- [46] Jane Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications*, 7(6):12–27, 1987.

- [47] Jane Wilhelms. Dynamic experiences. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 265–279. Morgan-Kaufmann, San Mateo, CA, 1991.
- [48] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, 1988.
- [49] Andrew Witkin, Kurt Fleisher, and Alan Barr. Energy constraints on parameterized models. *Computer Graphics*, 21(3):225–232, 1987.
- [50] David Zeltzer. Motor control techniques for figure animation. *IEEE Computer Graphics and Applications*, 2(9):53–59, Nov. 1982.
- [51] David Zeltzer. Knowledge-based animation. In N. I. Badler and J. K. Tsotsos, editors, *Motion: Representation and Perception*, pages 318–323. Elsevier, North Holland, New York, NY, 1986.
- [52] David Zeltzer. Task-level graphical simulation: Abstraction, representation, and control. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 3–33. Morgan-Kaufmann, San Mateo, CA, 1991.
- [53] David Zeltzer, S. Pieper, and D. Sturman. An integrated graphical simulation platform. In *Proc. Graphics Interface '89*, 1989.
- [54] Jianmin Zhao and Norman I. Badler. Real time inverse kinematics with joint limits and spatial constraints. Technical Report MS-CIS-89-09, Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1989.

Geometry for Computer Graphics

Mike Bailey

University of California at San Diego

San Diego Supercomputer Center

mjb@sdsc.edu



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

**Geometry
is
Our
Friend**

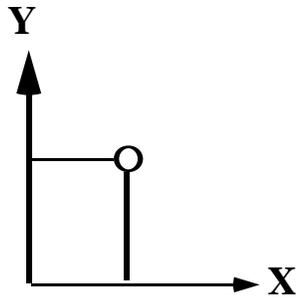
(no, really ...)



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

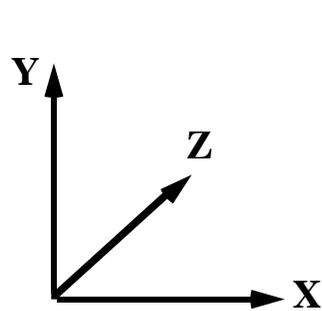
2D Coordinate System



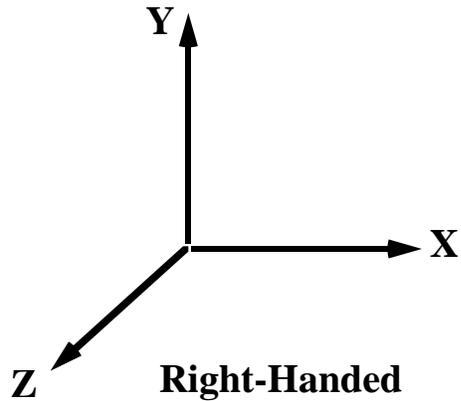
SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

3D Coordinate Systems



Left-Handed



Right-Handed



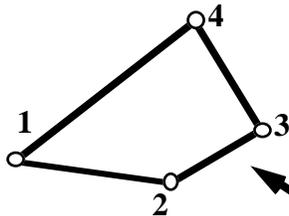
SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Geometry vs. Topology

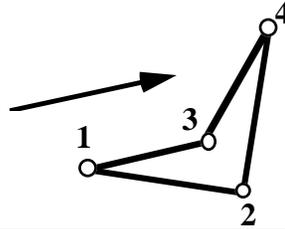
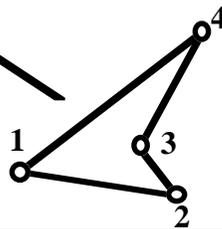
Geometry:

Where things are (e.g., coordinates)



Topology:

How things are connected



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Geometry: Points

2D or 3D

Not necessarily circular dots – can be any “marker”

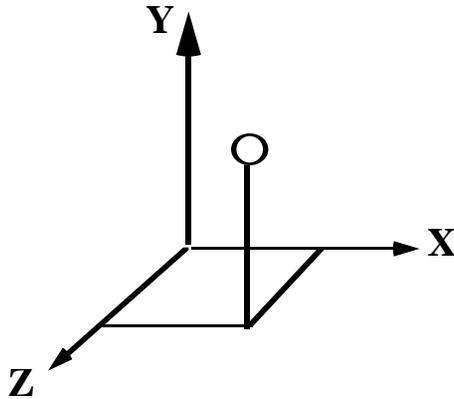
The string of points is sometimes referred to as a “polymarker”



SAN DIEGO SUPERCOMPUTER CENTER

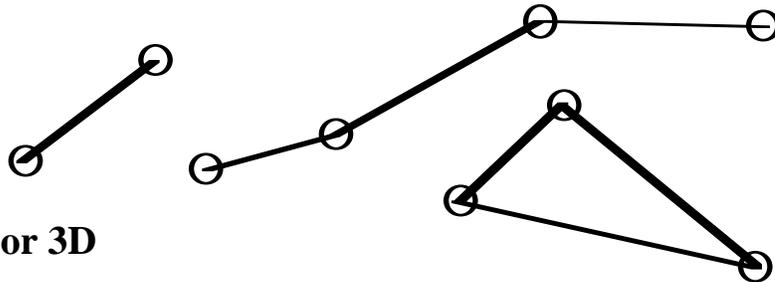
A National Center for Computational Science & Engineering

3D Points



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Lines



2D or 3D

Defined as a list of points

Topology varies

Sometimes referred to as a “polyline”



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Why “ $y=mx+b$ ” Isn't Good for Graphics Applications Software



- Cannot represent vertical lines ($m = \infty$)
- Can only represent infinite lines, not finite line segments
- Can only represent 2D lines, not 3D



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Parametric Line Equation



$$x = X1 + t * (X2 - X1)$$

$$y = Y1 + t * (Y2 - Y1)$$

$$z = Z1 + t * (Z2 - Z1)$$

$$0.0 \leq t \leq 1.0$$



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Can Also Be Thought of As a Blending Function



$$x = (1-t) * X1 + t * X2$$

$$y = (1-t) * Y1 + t * Y2$$

$$z = (1-t) * Z1 + t * Z2$$



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

$$0.0 \leq t \leq 1.0$$

Linear Blending Shows Up in a Lot of Computer Graphics Applications

You can linearly blend any two quantities with:

$$q = Q1 + t * (Q2 - Q1)$$

or, if you'd prefer:

$$q = (1 - t) * Q1 + t * Q2$$

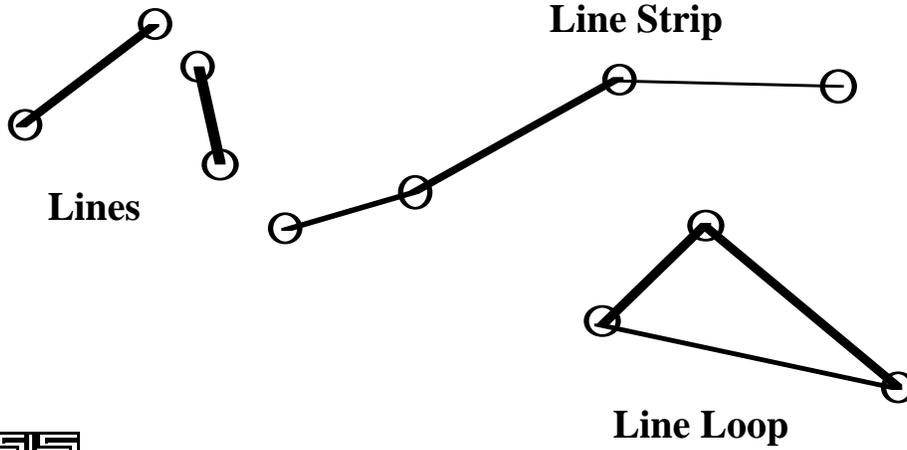
color, shape, location, angle, scale factors, ...



SAN DIEGO SUPERCOMPUTER CENTER

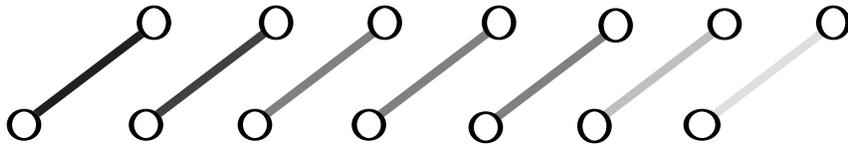
A National Center for Computational Science & Engineering

Line Topologies



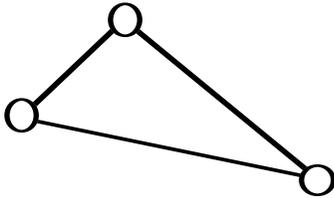
SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Line Patterns: Stipples



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Polygons

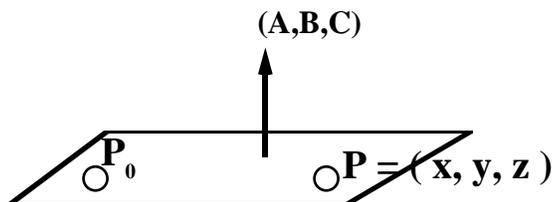


- Planar
- Defined as a closed sequence of points
- 2D or 3D



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Sidebar: What is “Planar?”



above

If the point P is on the plane, then:

below

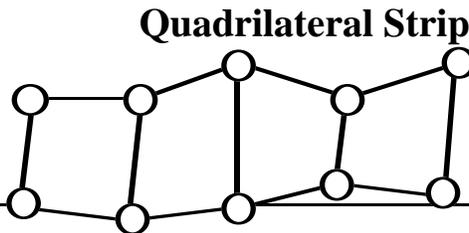
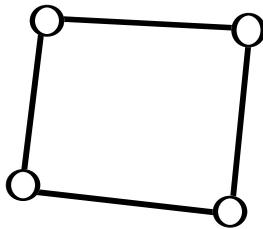
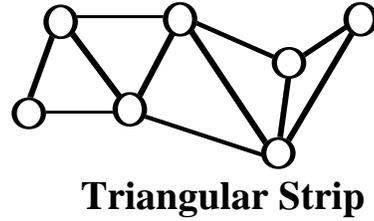
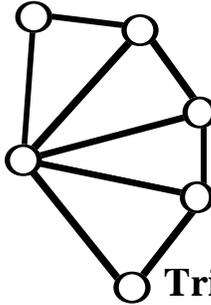
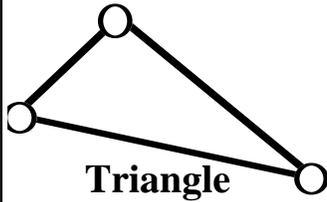
$$Ax + By + Cz - (Ax_0 + By_0 + Cz_0) = 0$$

>
<



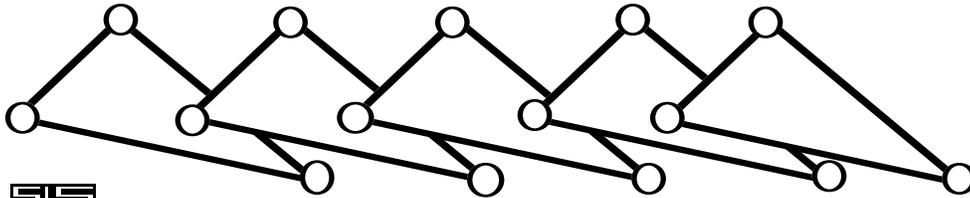
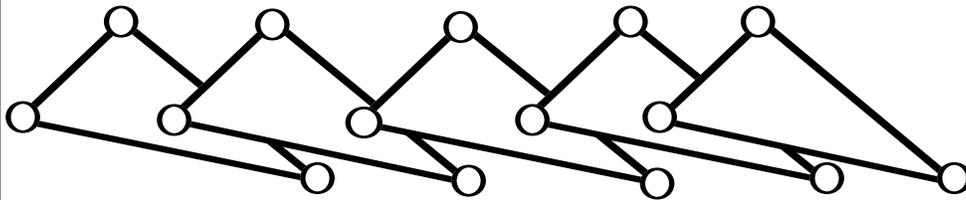
SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Some Special Polygon Topologies



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Polygon Patterns: Stipples



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Polygon Patterns: Color Interpolation



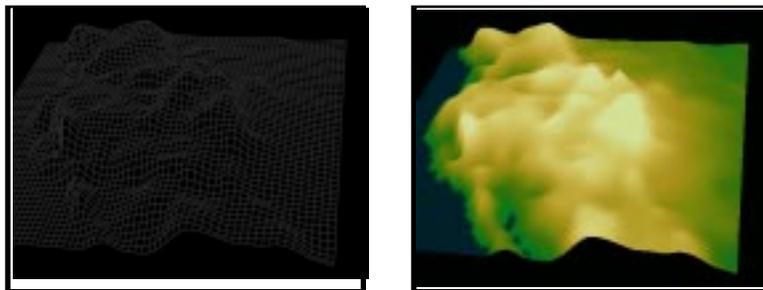
Referred to as *Smooth Shading*,
or *Gouraud Shading*



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

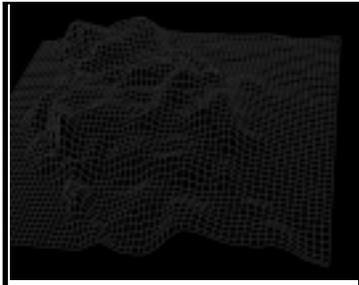
Polygon Patterns: Color Interpolation



SAN DIEGO SUPERCOMPUTER CENTER

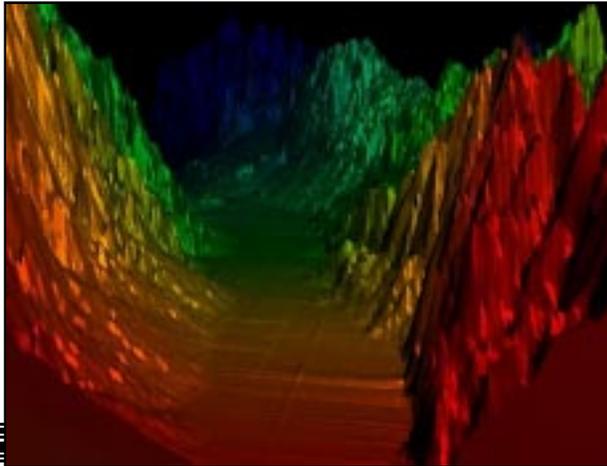
A National Center for Computational Science & Engineering

Polygon Patterns: Texture Mapping



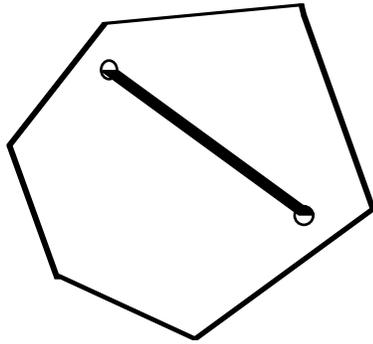
SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Texture Mapping: Automatically-Generated Textures

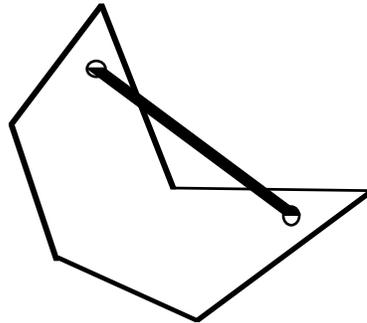


SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Convex vs. Concave



Convex



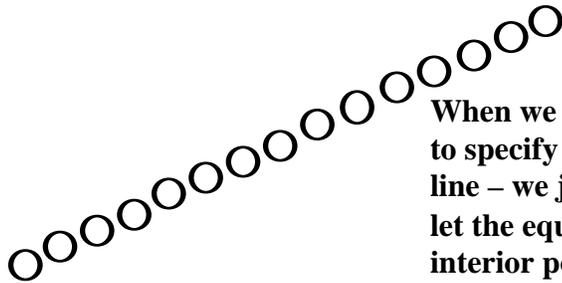
Concave



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Higher Order Geometry



When we draw a line, we do not need to specify all pixel points along the line – we just give the endpoints and let the equation determine the interior points

Can we do the same with other curve and surface types?

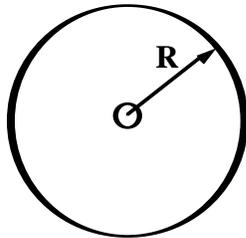


SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Conics

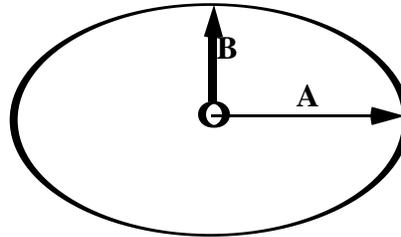
Circle



$$x = R\cos\theta$$

$$y = R\sin\theta$$

Ellipse



$$x = A\cos\theta$$

$$y = B\sin\theta$$

Parabola, Hyperbola, •••

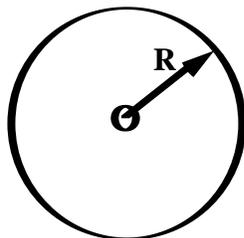


SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

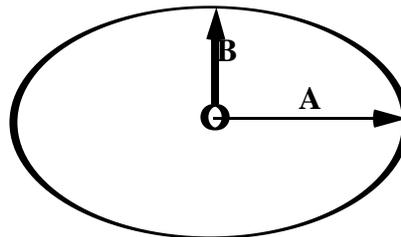
$$0.0 \leq t \leq 1.0$$

It is often handy to think of the independent parameter as consistently varying from 0.0 to 1.0



$$x = R\cos(360t)$$

$$y = R\sin(360t)$$



$$x = A\cos(360t)$$

$$y = B\sin(360t)$$

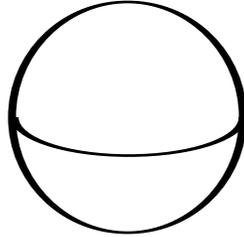


SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Quadrics

Sphere



Ellipsoid, Paraboloid, Hyperboloid, [Torus,] •••



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Arbitrary Curves



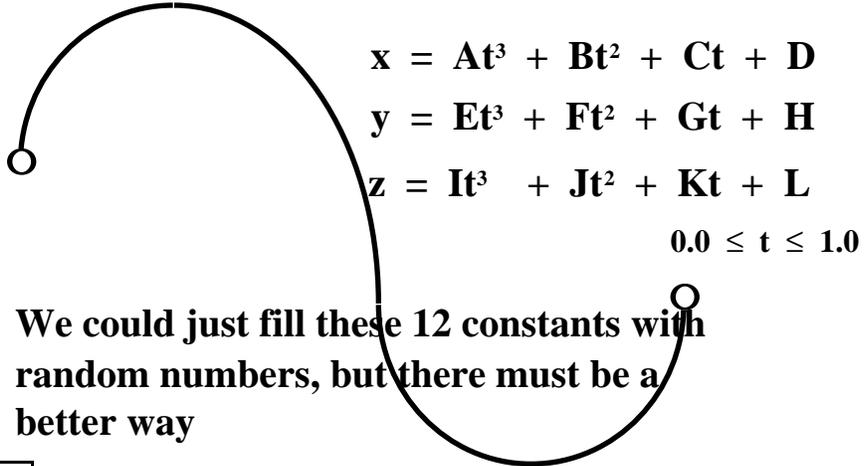
How do we control what goes on in here?



SAN DIEGO SUPERCOMPUTER CENTER

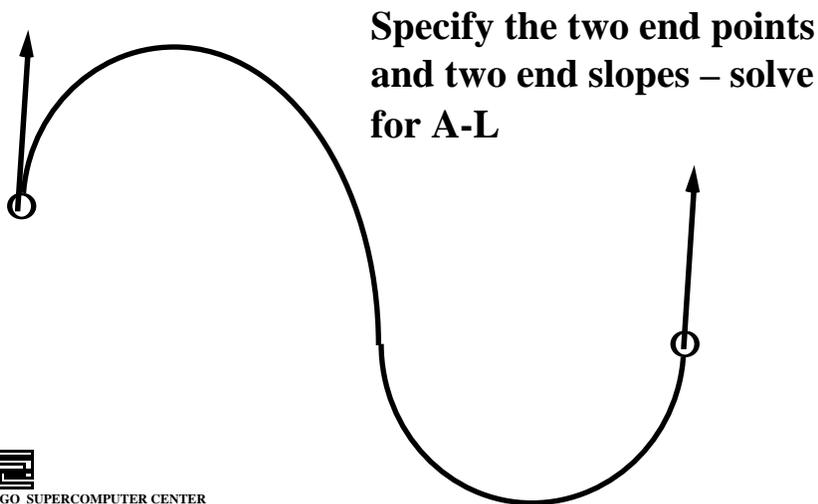
A National Center for Computational Science & Engineering

Cubic Curves



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

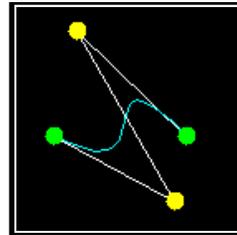
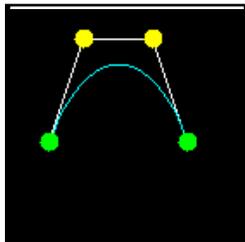
Hermite, or Coons, Cubic Curve



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Bézier Cubic Curves

Specify the two end points
and two “control points” –
solve for A-L



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

BiCubic Surfaces

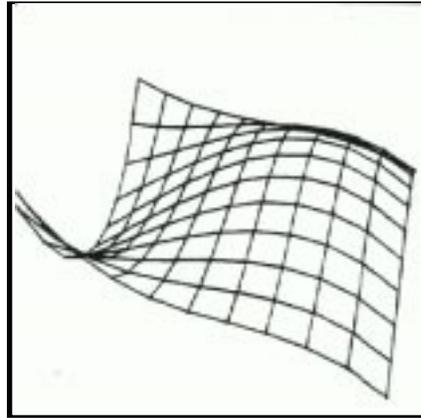
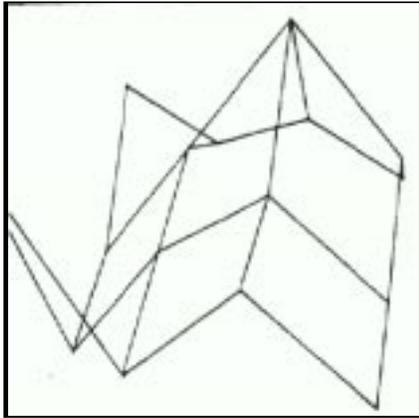
$$\mathbf{X}(s,t) = \begin{aligned} & \mathbf{A}s^3t^3 + \mathbf{B}s^3t^2 + \mathbf{C}s^3t + \mathbf{D}s^3 + \\ & \mathbf{E}s^2t^3 + \mathbf{F}s^2t^2 + \mathbf{G}s^2t + \mathbf{H}s^2 + \\ & \mathbf{I}st^3 + \mathbf{J}st^2 + \mathbf{K}st + \mathbf{L}s + \\ & \mathbf{M}t^3 + \mathbf{N}t^2 + \mathbf{O}t + \mathbf{P} \end{aligned}$$

$$0.0 \leq s,t \leq 1.0$$



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

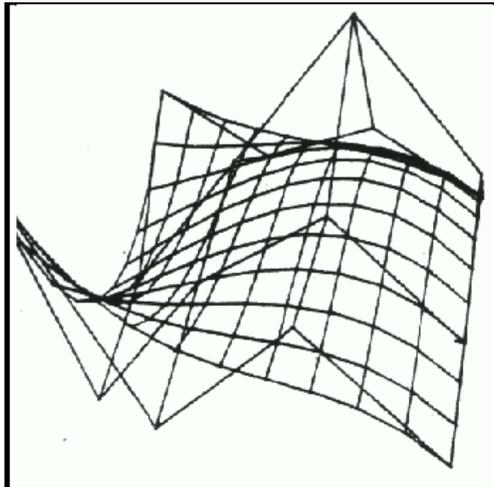
Bézier Bicubic Surfaces



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Bézier Bicubic Surfaces



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Transformations

**Mathematical equations
to change an object's coordinates**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Linear Equations

$$x' = Ax + By + Cz + D$$

$$y' = Ex + Fy + Gz + H$$

$$z' = Ix + Jy + Kz + L$$

Transform the geometry – leave the topology as is



SAN DIEGO SUPERCOMPUTER CENTER

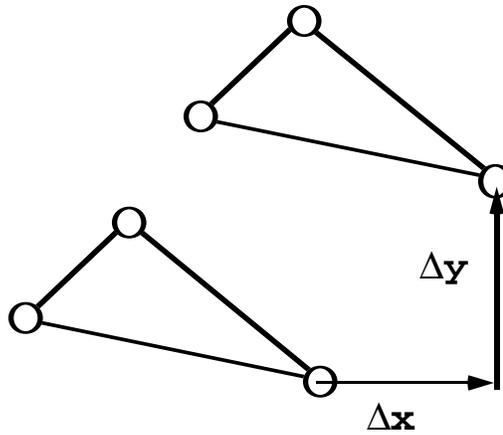
A National Center for Computational Science & Engineering

Translation

$$x' = x + \Delta x$$

$$y' = y + \Delta y$$

$$z' = z + \Delta z$$



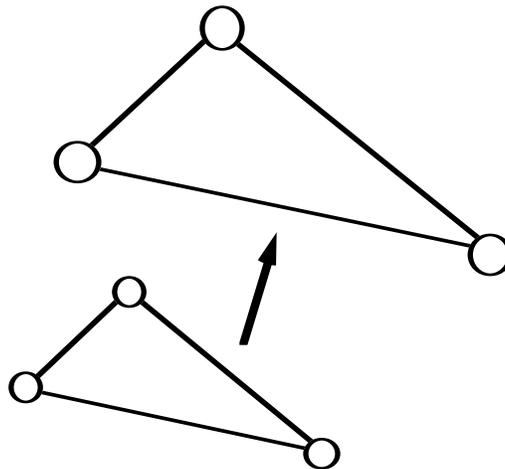
SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Scaling

$$x' = x * Sx$$

$$y' = y * Sy$$

$$z' = z * Sz$$

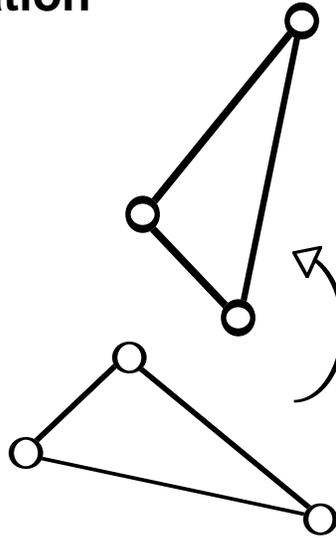


SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

2D Rotation

$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta$$



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Linear Equations in Matrix Form

$$x' = Ax + By + Cz + D$$

$$y' = Ex + Fy + Gz + H$$

$$z' = Ix + Jy + Kz + L$$

$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Identity Matrix

$$x' = x$$

$$y' = y$$

$$z' = z$$

$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

[I] signifies that “Nothing has changed”

Matrix Inverse

$$[M] \bullet [M]^{-1} = [I]$$

$$[M] \bullet [M]^{-1} = \text{“Nothing has changed”}$$

“Whatever [M] does, [M]⁻¹ undoes”



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Translation

$$x' = x + \Delta x$$

$$y' = y + \Delta y$$

$$z' = z + \Delta z$$

$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Scaling

$$x' = x * Sx$$

$$y' = y * Sy$$

$$z' = z * Sz$$

$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

2D Rotation

$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta$$

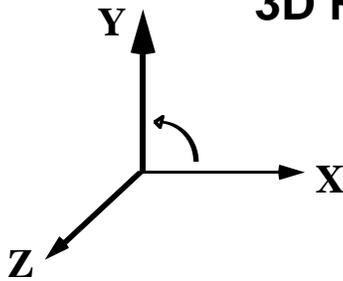
$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

3D Rotation About Z



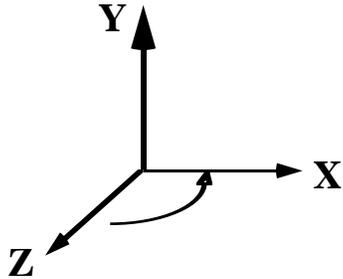
$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

3D Rotation About Y

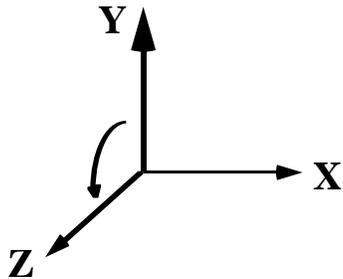


$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

3D Rotation About X



$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Compound Transformations

Write it

$$\begin{matrix} x' \\ y' \\ z' \\ 1 \end{matrix} = \left([T \ +A, +B] * \left([R \ \theta] * \left([T \ -A, -B] * \begin{matrix} x \\ y \\ z \\ 1 \end{matrix} \right) \right) \right)$$

Say it

SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Matrix Multiplication is Not Commutative

Translate, then rotate

SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Matrix Multiplication is Associative

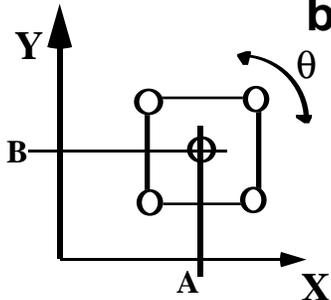
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \left([T \ +A, +B] * \left([R \ \theta] * \left([T \ -A, -B] * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \right) \right) \right)$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \left(\left([T \ +A, +B] * \left([R \ \theta] * [T \ -A, -B] \right) \right) * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \right)$$



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Can Multiply All Geometry by One Matrix !

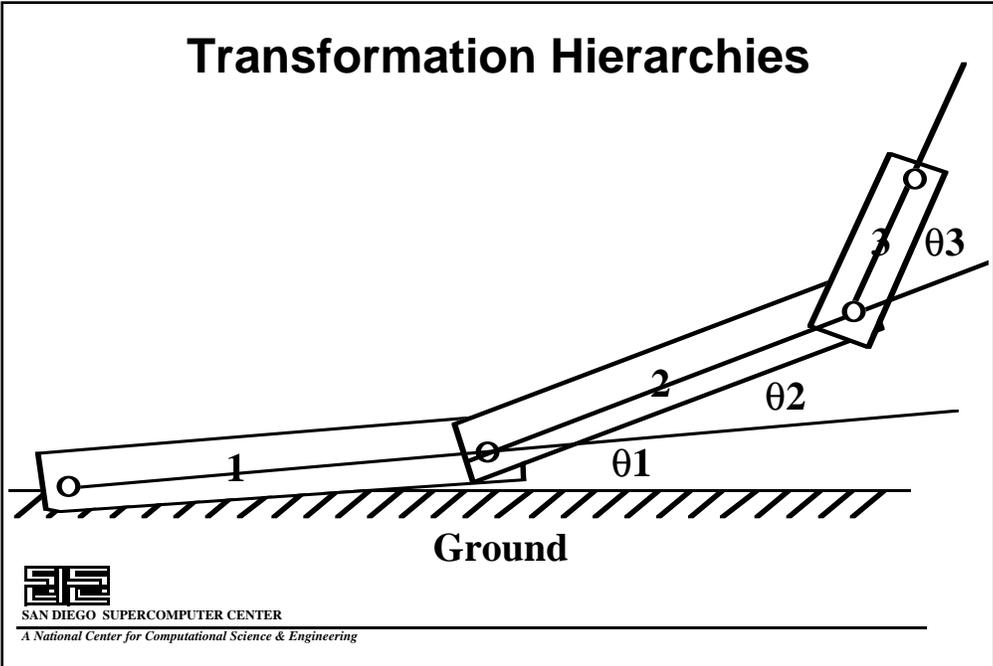
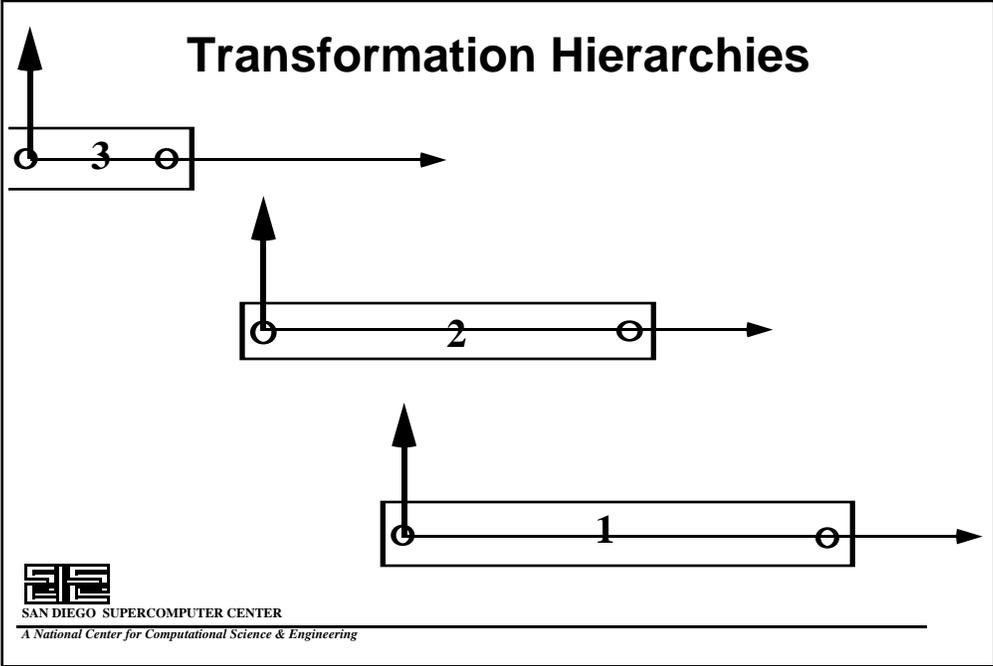


$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \left(\left(\mathbf{M} \right) * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \right)$$



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Hardware can do this very quickly!



Positioning Part #1 With Respect to Ground

1. Rotate by θ_1
2. Translate by $\Delta_{1/G}$

Write it

$$\begin{aligned}
 [M_{3/G}] &= [T_{1/G}] * [R_{\theta_1}] \\
 &= [M_{1/G}]
 \end{aligned}$$

Say it



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Positioning Part #2 With Respect to Ground

1. Rotate by θ_2
2. Translate the length of part 1
3. Rotate by θ_1
4. Translate by $\Delta_{1/G}$

Write it

$$\begin{aligned}
 [M_{2/G}] &= [T_{1/G}] * [R_{\theta_1}] * [T_{2/1}] * [R_{\theta_2}] \\
 &= [M_{1/G}] * [M_{2/1}]
 \end{aligned}$$

Say it



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Positioning Part #3 With Respect to Ground

1. Rotate by θ_3
2. Translate the length of part 2
3. Rotate by θ_2
4. Translate the length of part 1
5. Rotate by θ_1
6. Translate by $\Delta_{1/G}$

Write it

$$\begin{aligned} [M_{3/G}] &= [T_{1/G}] * [R_{\theta_1}] * [T_{2/1}] * [R_{\theta_2}] * [T_{3/2}] * [R_{\theta_3}] \\ &= [M_{1/G}] * [M_{2/1}] * [M_{3/2}] \end{aligned}$$



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Say it

Application Programming Interfaces (APIs)

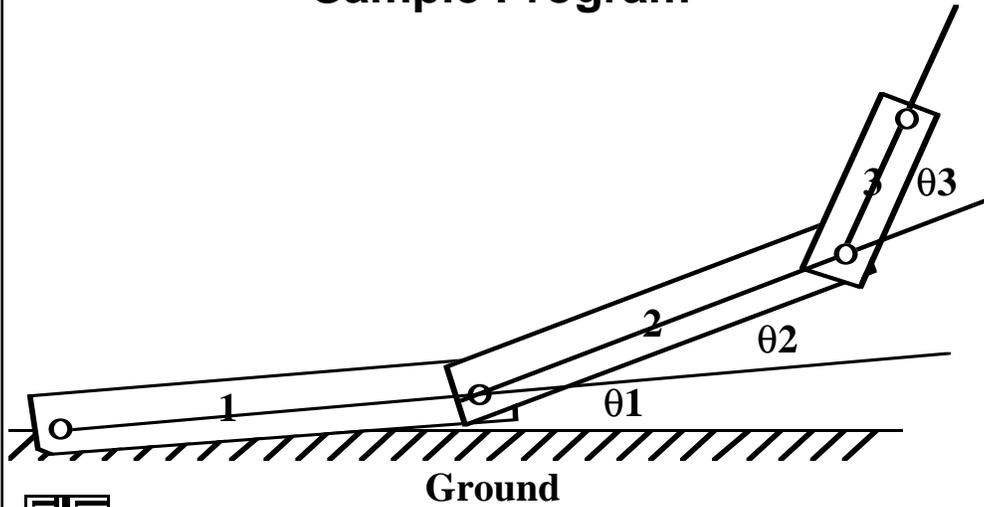
The way the application gains access to the graphics

- OpenGL
- Direct3D
- VRML
- Java 3D
- Fahrenheit



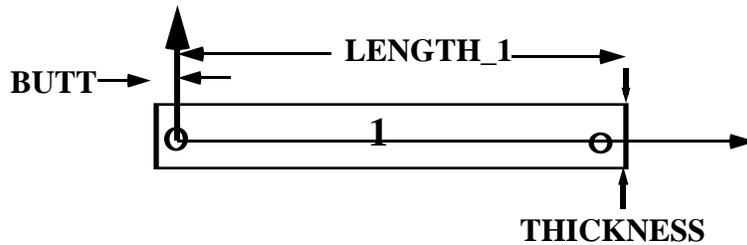
SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Sample Program



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Sample Program



```
DrawLinkOne( )
{
    glBegin( GL_LINE_LOOP );
    glVertex2f( -BUTT, -THICKNESS/2 );
    glVertex2f( LENGTH_1, -THICKNESS/2 );
    glVertex2f( LENGTH_1, THICKNESS/2 );
    glVertex2f( -BUTT, THICKNESS/2 );
    glEnd();
}
```



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Sample Program

```

DrawMechanism(  $\theta_1$ ,  $\theta_2$ ,  $\theta_3$  )
    float  $\theta_1$ ,  $\theta_2$ ,  $\theta_3$ ;
    {
        glPushMatrix();
        glRotatef(  $\theta_1$ , 0., 0., 1. );
        glIndexi( RED );
        DrawLinkOne();

        glTranslatef( LENGTH_1, 0., 0. );
        glRotatef(  $\theta_2$ , 0., 0., 1. );
        glIndexi( GREEN );
        DrawLinkTwo();

        glTranslatef( LENGTH_2, 0., 0. );
        glRotatef(  $\theta_3$ , 0., 0., 1. );
        glIndexi( BLUE );
        DrawLinkThree();
    }
    glPopMatrix();

```



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Sample Program

Where in the
window to
display (pixels)

Viewing Info:
field of view
angle, x:y aspect
ratio, near, far

Whatever
interaction is
being used

Eye position

```

glViewport( 100, 100, 500, 500 );

glMatrixMode( GL_PROJECTION );
glLoadIdentity( );
gluPerspective( 90., 1.0, 1., 10. );

glMatrixMode( GL_MODELVIEW );
glLoadIdentity( );

done = FALSE;
while( ! done )
    {
        << Determine  $\theta_1, \theta_2, \theta_3$  >>
        glPushMatrix();
        gluLookAt( eyex, eyey, eyez,
                  centerx, centery, centerz,
                  upx, upy, upz );
        DrawMechanism(  $\theta_1$ ,  $\theta_2$ ,  $\theta_3$  );
        glPopMatrix();
    }

```



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Good Geometry References

Foley, Dam, Feiner, Hughes, and Phillips, *Introduction to Computer Graphics*, Addison-Wesley, 1993.

Rogers and Adams, *Mathematical Elements for Computer Graphics*, McGraw-Hill, 1989.

Taylor, *The Geometry of Computer Graphics*, Wadsworth & Brooks/Cole, 1992.

Farin, *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, 1990.



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Good Geometry References

Bartels, Beatty, Barsky, *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*, Morgan-Kaufmann, 1987.

Hoffman, *Geometric & Solid Modeling*, Morgan Kaufmann, 1989.

Mortenson, *Geometric Modeling*, John Wiley & Sons, 1985.

Faux and Pratt, *Computational Geometry for Design and Manufacture*, Ellis-Horwood, 1979.

Graphics Gems 1-5, Academic Press.



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Graphical Input Devices

Mike Bailey

University of California at San Diego

San Diego Supercomputer Center

mjb@sdsc.edu



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Logical Input Device Types

- **Choice**
- **Keyboard**
- **Valuators**
- **Locators**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Choice

- **Return a choice that has been made**
- **Examples: function keypad, button box, foot switch**
- **Often provide sensorial feedback: lights, clicks, feel, . . .**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Button Box



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Keyboard

- **Returns keys with specific meanings**
- **Letters, numbers, etc.**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Valuators

- **Return a value for something**
- **Example: knobs**
- **Can usually specify gain, minimum, and maximum**
- **All locators can also double as valuators**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Dial Box



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Locators

- **Return the location of the screen cursor**
- **Examples: mouse, tablet, trackball, touchscreen**
- **Display-to-Input ratio**



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Locator Display-to-Input Ratio

DTI Ratio: the amount of cursor movement on the screen divided by the amount of hand movement

Large: good for speed

Small: Good for accuracy

Sometimes called “Gain”



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Ways to Read an Input Device

- **Sampling:** What is its input *right now* ?
- **Event-based:** Wait until the user does something



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

3D Input devices

- **Read a 3D position**
- **Returns 3 numbers to the program: an (x,y,z) triple**
- **Some also return 3 more numbers to the program: three rotation angles**
- **Examples: digitizer, spaceball, glove**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

3D Input Devices



SAN DIEGO SUPERCOMPUTER CENTER

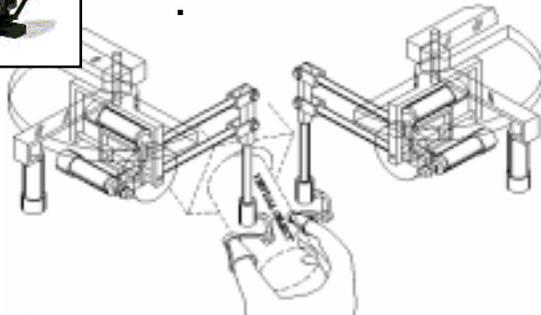
A National Center for Computational Science & Engineering

3D Input Devices



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Force Feedback in 3D



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Force Feedback in 3D



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Force Feedback in 2D



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Computer Graphics on the World Wide Web

Mike Bailey

University of California San Diego
San Diego Supercomputer Center

mjb@sdscc.edu



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Image Files

- GIF
- JPEG
- PNG



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

GIF Image Files

- **Graphics Interchange Format**
- **Up to 8 bits with CLT**
- **Can be interleaved**
- **Can have a transparent background**
- **Can be compressed**
- **Can have multiple frames, an interframe delay time, and a repeat count**



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

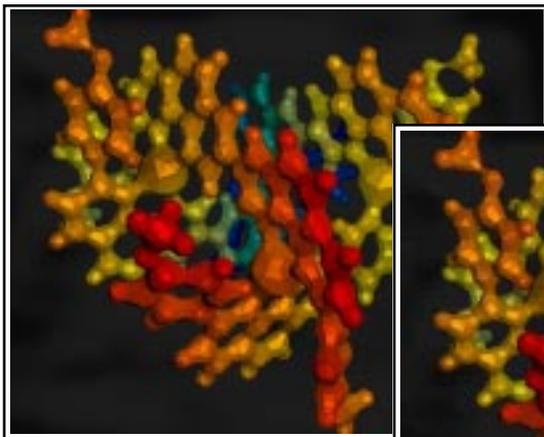
JPEG Image Files

- **Joint Photographic Experts Group**
- **Image can be 24-bit color**
- **Can be arbitrarily compressed**
- **www.jpeg.org**
- **The following example shows a 1080x852 24-bit image (uncompressed = 2,760,480 bytes)**

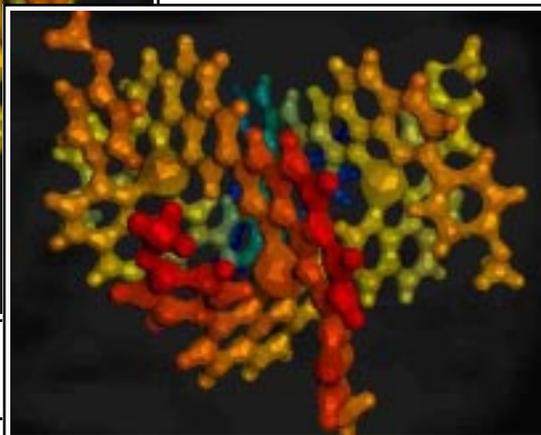


SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

174,628 Bytes **JPEG Compression**

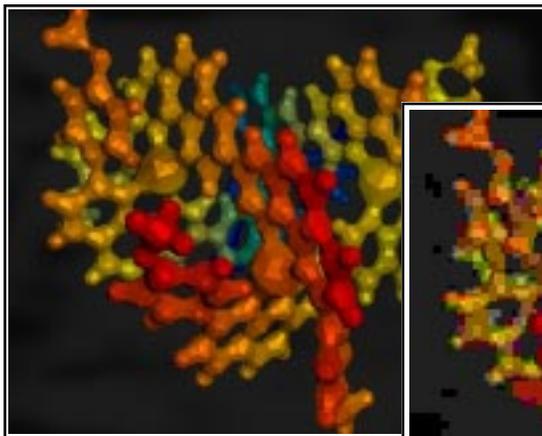


16,167 Bytes

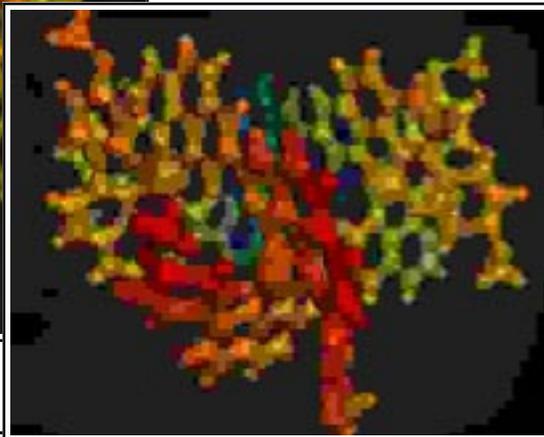


SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

174,628 Bytes **JPEG Compression**



5,311 Bytes



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

PNG

- **Portable Network Graphics**
- **Designed to replace GIF**
 - **variable transparency**
 - **2D interleaving**
 - **better compression**
- **www.cdrom.com/pub/png**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Animation Files

- **Animated GIF**
- **MPEG**
- **Quicktime**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Animated GIF

- **GIF files can have multiple frames, an interframe delay time, and a repeat count**
- **One freeware package is WhirlGif:**
`www.danbbs.dk/~dino/whirlgif`



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

MPEG

- **Moving Picture Experts Group**
- **Codes video and audio**
- **Highly compressed**
- `www.mpeg.org`



SAN DIEGO SUPERCOMPUTER CENTER
A National Center for Computational Science & Engineering

Quicktime

- **Product of Apple Computer**
- **Codes video and audio**
- **Highly compressed**
- **`www.apple.com/quicktime`**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

VRML

- **Virtual Reality Modeling Language**
- **3D scene coded as an ASCII file**
- **Scene content (ie, geometry) defined in nodes**
- **Node attributes given as fields and values (eg, size, appearance, lighting properties)**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

VRML Example

```
Cylinder  
{
```

```
Cylinder  
{  
  radius    2.0  
  height    4.0  
}
```



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

VRML Example

```
#VRML 2.0 utf8  
Shape  
{  
  appearance  
  Appearance  
  {  
    material  
    Material { }  
  }  
  geometry  
  Cylinder  
  {  
    radius  2.0  
    height  4.0  
  }  
}
```



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

For More VRML Information, See:

`vrml.sdsc.edu`



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Java 3D

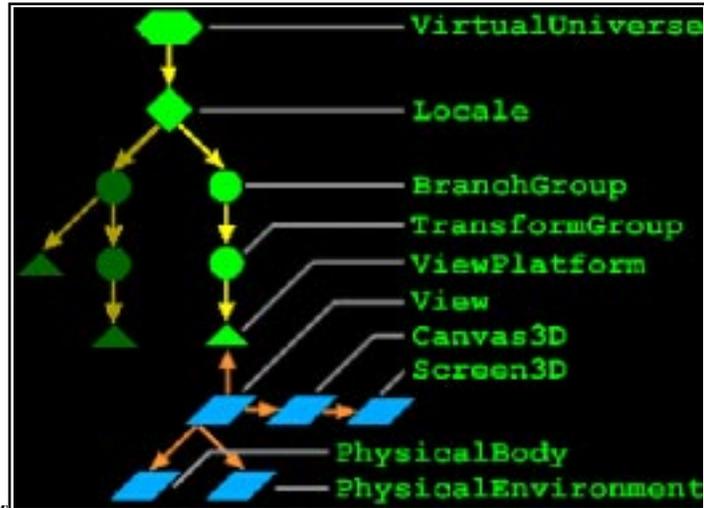
- **3D Graphics API for Java**
- **Primary mode is a 3D scene-graph display list**
- **Display list is highly optimized**
- **View model is separate from the scene model**



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

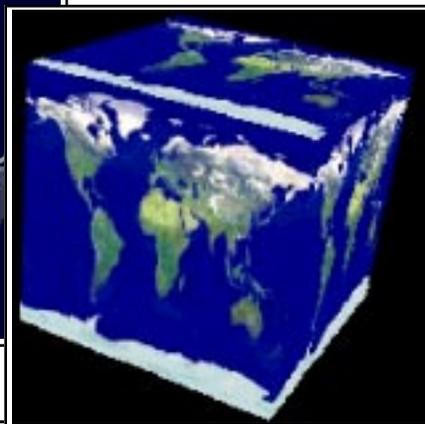
Java 3D Scene Graph



SAN DIEGO SUPERCOM

A National Center for Computational Science & Engineering

Java 3D Examples



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

For More Java 3D Information, See:

www.sun.com/desktop/java3d

java.sun.com/products/java-media/3D

java3d.sdsc.edu



SAN DIEGO SUPERCOMPUTER CENTER

A National Center for Computational Science & Engineering

Virtual Reality

By Olin Lathrop, Updated April 1999

What is Virtual Reality?

Virtual Reality is a bad name for a good idea. Throughout the relatively short history of computer graphics, engineers have strived to develop more realistic, responsive, and immersive means for the human to interact with the computer. What is often called *virtual reality* (or simply *VR*) is the latest in that progression.

The name *Virtual Worlds* is being used by some in an attempt to be more realistic. Unfortunately this has not yet caught on in a major way. You are still much more likely to hear the term *Virtual Reality*, so that's what I'll use in this introduction to avoid confusing you with non-standard terms.

While there is no one definition of VR everyone will agree on, there is at least one common thread. VR goes beyond the flat monitor that you simply look at, and tries to immerse you in a three dimensional visual world. The things you see appear to be in the room with you, instead of stuck on a flat area like a monitor screen. As you might imagine, there are a number of techniques for achieving this, each with its own tradeoff between degree of immersion, senses involved beyond sight, computational requirements, physical constraints, cost, and others.

Some VR Systems

We'll start simple and build step by step to more fancy systems. This will help you understand how we got to where we are, and better appreciate the wide range of solutions with their various tradeoffs.

Stereo Viewing

An important aspect of VR is that the things you look at are presented in *three dimensions*. Humans see the world around them in 3D using many different techniques. Probably the most powerful technique for nearby (a few tens of meters) objects is called *stereoscopic*.

In stereoscopic 3D perception, we use the fact that we have two eyes that are set some distance apart from each other. When we look at a nearby object we measure the difference in angle from each eye to the object. Knowing this angle, the distance between the eyes, and a bit of trigonometry, we compute the distance to the object. Fortunately this is done for us sub-consciously by an extremely sophisticated image processor in our brain. We simply perceive the end result as objects at specific locations in the space around us.

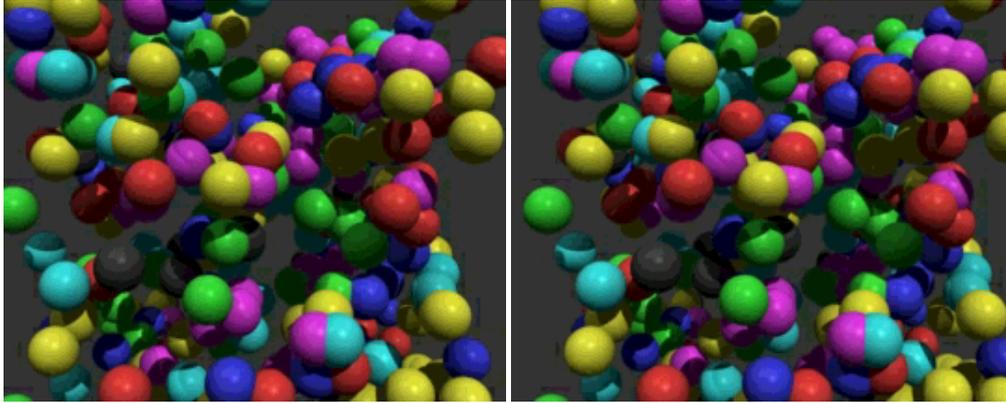


Figure 1 - A Stereo Pair of Images

Cross your eyes so that the two images overlap as one image in the center. This may be easier if you move back from the page a bit. After a while, your eyes will lock and re-focus on the center image. When that happens, you will see the 3D layout of the spheres pop out at you. Don't feel bad if you can't see it. I've found that roughly 1/3 of the people can see it within a minute, another 1/3 can see it with practise, and the remaining 1/3 seem to have a hard time.

Figure 1 shows stereo viewing in action. When you cross your eyes as directed, your right eye sees the left image, and your left eye sees the right image. For those of you that can see it, the effect is quite pronounced, despite the relatively minor differences between the two images.

So, one way to make a 3D computer graphics display is to render two images from slightly different eye points, then present them separately to each eye. Once again there are several ways of achieving this.

Shutter Glasses

Probably the simplest way of displaying stereo computer images is by using the existing monitor. Suppose the display alternates rapidly between the left and right eye images. We could then make sure each eye only saw the image intended for it by opening a shutter in front of the eye when its image is being displayed. The shutters would have to be synchronized to the display.

This is exactly what shutter glasses are. They typically use electronic shutters made with liquid crystals. Another variation places the shutter over the monitor screen. Instead of blocking or unblocking the light, this shutter changes the light polarization between the left and right eye images. You can now wear passive polarizing glasses where the polarizer for each eye only lets thru the image that was polarized for that eye.

While this form of stereo viewing can provide good 3D preception and is suitable for many tasks, I don't personally consider this VR yet. The image or 3D scene is still "stuck" in the monitor. You are still looking at the picture in a box, instead of being in the scene with the objects you are viewing.



Figure 2 - Shutter Glasses and Controller
CrystalsEyes product by StereoGraphics. This image was swiped from their web page.

Head Mounted Display

Another way to present a separate image to each eye is to use a separate monitor for each eye. This can be done by mounting small monitors in some sort of head gear. With the right optics, the monitors can appear large and at a comfortable viewing distance. This setup is usually referred to as a *head mounted display*, or HMD for short.

I don't think of this by itself as VR yet either, although we're getting closer. You can have a reasonable field of view with great latitude in the placement of 3D objects, but the objects appear to move with your head. That's certainly not what happens when you turn your head in Real Reality.



Figure 3 - Head Mounted Display in Use
The head mounted display is a Virtual Research VR4000. This image was swiped from their web page.

Head Tracking

What if the computer could sense the position and orientation of your head in real time? Assuming we have a sufficiently fast computer and a head mounted display (that's where the position/orientation sensor is hidden), we could re-render the image for each eye in real time also, taking into account the exact eye position. Objects could then be defined in a fixed space. As you moved your head around, the eye images would update to present the illusion of a 3D object at a fixed location in the room, just like real objects. Of course, the objects themselves can be moving too. But the important thing is that they are moving within some fixed frame that you can also move around in.

Now we've reached the minimum capabilities I'm willing to call VR.



Figure 4 - Position and Orientation Sensors

UltraTrak product by Polhemus. The position and orientation of each of the sensors on the table are reported to the computer in real time. The data is relative to the large ball on top of the unit, which contains three orthogonal magnetic coils. Individual sensors can be imbedded in a head mounted display, fixed to various body parts, or mounted on other object that move around. This image was swiped from the Polhemus web page.

Hand Tracking

But wait, there's more. We can use more motion sensors and track the position and orientation of other objects, like your fingers, for example. Just like a mouse or joystick can be used to control a program, your finger actions could be used to control a program. This

might take the form of pushing virtual menu buttons, or maybe grabbing an object and moving it around with your hand. The possible interactions are virtually boundless, limited mostly by the software designer's imagination.

Hand and finger position and orientation is typically achieved by wearing a special glove that has a position sensor on it and can sense the angles of your finger joints. This can be taken a step further by wearing a whole suite with imbedded joint angle sensors.



Figure 5 - Glove that senses joint angles

This is the CyberGlove by Virtual Technologies. It reports various joint angles back to the computer. This image was swiped from a Virtual Technologies web page.

Force Feedback

So far we can be in the same space with the objects we're viewing and interact with them thru hand motions, but we can't feel them. That's where force feedback comes in. This is also referred to as *haptic feedback*. Suppose the special glove (or body suit) you were wearing could not only sense joint angles, but also had actuators that could push back at you. With some clever software and fast computers, the actuators could present the illusion of

hard objects at particular locations. You can now not only see it in 3D, walk around it, control it, but also bump into it.

Note that force feedback is currently limited to "pushing back" to simulate the existence of an object. It does not provide other parts of what we call tactile feel, like texture, temperature, etc.

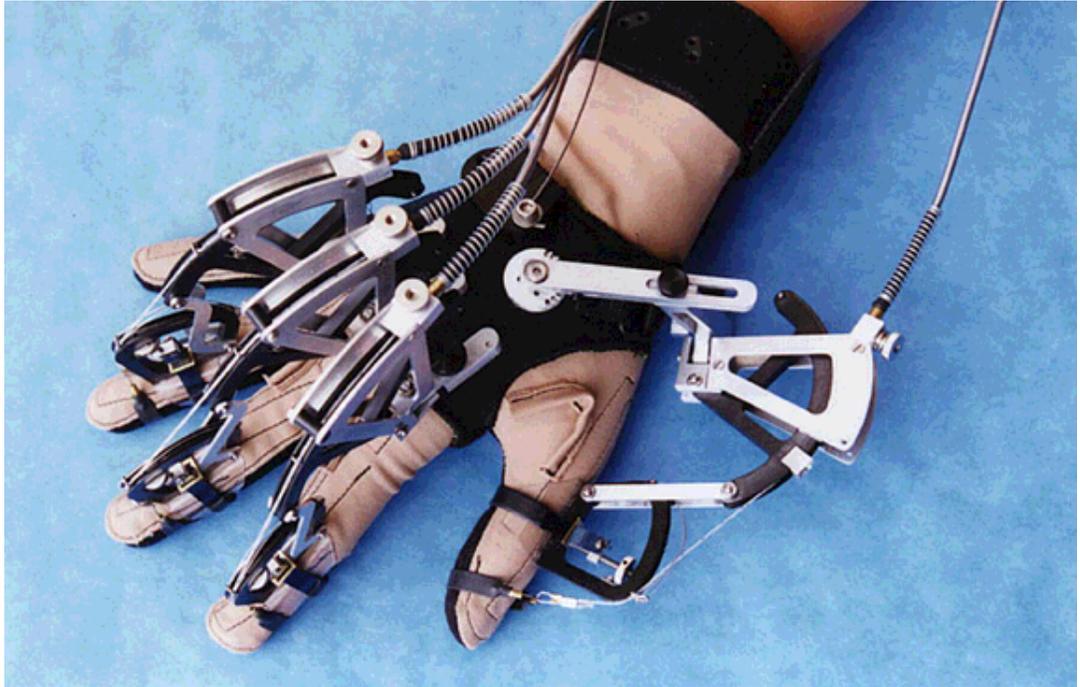


Figure 6 - Haptic Feedback Glove

No, this isn't some medieval torture device. It's the Cybergrasp product by Virtual Technologies. The cables and pulleys on the outside of the glove can be used to "push back" at the operator under computer control. This can be used, among other things, to allow the wearer to feel the presence of objects in the virtual world.

The CAVE

Instead of using a head mounted display, imagine a room where output of computer displays is projected onto the walls. The projected images are in stereo by rapidly alternating between the two eye images. You stand somewhere near the middle and wear shutter glasses for a 3D effect.

This concept was first developed in 1991 at the Electronic Visualization Lab of the University of Illinois at Chicago. Several other CAVE systems have since been set up at other sites.

Applications

I'm just going to briefly mention a few areas VR technology is used.

Entertainment

This is definitely the biggest market, and is the main force for driving down prices on VR hardware. You can be in a computer game with computer generated players and/or other real players.



Figure 7 - Virtual Reality Gaming

An Ascension Technology SpacePad motion tracker is being used in conjunction with a head mounted display and hand buttons to implement a gaming station. This image was swiped from the Ascension Technology web page.

Augmented Reality

Imagine a VR head mounted display as we've discussed, but the display doesn't block out the regular view, it's just superimposed on it. Imagine walking around a building and "seeing" inside the walls to the wiring, plumbing, and structure. Or, seeing the tumor inside a patient's head as you hack away at it.

Training

VR is already being used in to teach people how to use expensive equipment, or when the cost of a mistake in Real Reality is very high. For example, use of VR is getting more common in aircraft simulators used as part of an overall program to train pilots. The benefits are also substantial for military applications for obvious reasons.

Remote Robotics

This is another "real" application that is gaining much attention. Suppose you had a robot that had arms and hands modeled after those of humans. It could have two video cameras where we have eyes. You could be wearing a head mounted display and see what the robot sees in real time. If your head, arm, and hand motions are sensed and replicated in the robot,

for many applications you could be where the robot is without really being there.

This could be useful and worth all the trouble in situations where you can't physically go, or you wouldn't be able to survive. Examples might be in hostile environments like high radiation areas in nuclear power plants, deep undersea, or in outer space. The robot also doesn't need to model a human exactly. It could be made larger and stronger to perform more physical labor than you could, or it might be smaller to fit into a pipe you couldn't. Remote robotics could also be a way to project a special expertise to a remote site quicker than bringing a person there. Some law enforcement and military applications also come to mind.

Distributed collaboration

VR is being employed to allow geographically distributed people to do more together than simply hear and see each other as allowed by telephone or videoconferencing.

For example, the military is using VR to create virtual battles. Each soldier participates from his own point of view in an overall simulation that may involve thousands of individuals. All participants need not be in physical proximity, only connected on a network.

This technology is still in its infancy, but evolving rapidly. I expect commercial applications of distributed collaboration to slowly gain momentum over the next several years.

Visualization

Scientists at NASA/Ames and other places have been experimenting with VR as a visualization research tool. Imagine being able to walk around a new aircraft design as it's in a simulated mach 5 wind tunnel. VR can be used to "see" things humans can't normally see, like air flow, temperature, pressure, strain, etc.



Figure 8 - Simulated picture of VR used in visualization

This picture tries to show us what the virtual world looks like to the two engineers. Some of this is wishful thinking, but it still does a good job of illustrating the concepts. Note that no finger position sensors or force feedback devices are apparent, even though the bottom engineer is selecting virtual menu buttons, and the other engineer seems to be resting his hand on the model. Also, anyone in the room not wearing a display would not be able to see the model at all. This image was swiped from the Virtual Research home page.

Problems

The current state of VR is far from everything we could imagine or want. A few of the open issues are:

Cost

This stuff is just too expensive for everyone to have one, and it's likely to stay that way for

quite a while.

What's it Good For?

I listed some application areas above, but note that none of them solve common everyday problems. While VR certainly has its application niches - and the number is steadily growing - it's hard to imagine how it can help the average secretary type a letter on a word processor.

Display Resolution

Head mounted displays need to be small and light else you get a sore neck real fast. Unfortunately, the display resolution is therefore limited. Most displays are only about 640 pixels across, which is a tiny fraction of what a normal human can see over the same angle of view.

Update Speed

Most VR displays are updated at 30 Herz (30 times per second). This requires a large amount of computation, just to maintain what looks and feels like "nothing is happening". The amount of computation required also depends on the scene complexity. VR is therefore limited to relatively "simple" scenes that can be rendered in 1/30 second or faster. This currently precludes any of the rendering methods that can provide shadows, reflections, transparency, and other realistic lighting effects. As a result, VR scenes look very "computer-ish".

The standard 30 Herz update rate comes from existing systems intended for displaying moving images on a fixed monitor or other screen. With VR, you also have to take into account how fast something needs to be updated as the user's head turns to present the illusion of a steady object. Apparently, that requires considerably more than 30 Herz for the full effect.

Links to Other VR Resources

- www.hut.fi/~then/vr.html Very nice collection of links to VR pages.
- www.cs.ualberta.ca/~graphics/MRvrhardware/MRvrhardware.html Lots of links to VR hardware companies.
- www.evl.uic.edu/EVL/VR/CAVE.overview.shtml - More info on the CAVE project.
- duchamp.arc.nasa.gov/papers/veei/veei.html - Much more extensive and scientific paper on Virtual Environments with many references.

Where to Find More Information on Computer Graphics

Mike Bailey

1. References

1.1 General

SIGGRAPH Online Bibliography Database:

<http://www.siggraph.org/publications/bibliography>

Jim Foley, Andy Van Dam, Steven Feiner, and John Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, 1990.

Jim Foley, Andy Van Dam, Steven Feiner, John Hughes, and Richard Phillips, *Introduction to Computer Graphics*, Addison-Wesley, 1993.

Edward Angel, *Interactive Computer Graphics: A Top-down Approach with OpenGL [1.0]*, Addison-Wesley, 1997.

Olin Lathrop, *The Way Computer Graphics Works*, John Wiley & Sons, 1997.

Andrew Glassner, *Graphics Gems*, Academic Press, 1990.

James Arvo, *Graphics Gems 2*, Academic Press, 1991.

David Kirk, *Graphics Gems 3*, Academic Press, 1992.

Paul Heckbert, *Graphics Gems 4*, Academic Press, 1994.

Alan Paeth, *Graphics Gems 5*, Academic Press, 1995.

David Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill, 1997.

SIGGRAPH Conference Final program.

1.2 Math and Geometry

Walter Taylor, *The Geometry of Computer Graphics*, Wadsworth & Brooks/Cole, 1992.

Gerald Farin, *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, 1996.

Gerald Farin and Dianne Hansford, *The Geometry Toolbox for Graphics and Modeling*, AK Peters, 1998.

Barrett O'Neil, *Elementary Differential Geometry*, Academic Press, 1997.

Joseph O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1996.

Christopher Hoffman, *Geometric & Solid Modeling*, Morgan Kaufmann, 1989.

Michael Mortenson, *Geometric Modeling*, John Wiley & Sons, 1985.

I.D. Faux and M.J. Pratt, *Computational Geometry for Design and Manufacture*, Ellis-Horwood, 1979.

Eric Stollnitz, Tony DeRose, and David Salesin, *Wavelets for Computer Graphics*, Morgan-Kaufmann, 1996.

Ronen Barzel, *Physically-Based Modeling for Computer Graphics*, Academic Press, 1992.

David Rogers and J. Alan Adams, *Mathematical Elements for Computer Graphics*, McGraw-Hill, 1989.

John Snyder, *Generative Modeling for Computer Graphics and Computer Aided Design*, Academic Press, 1992.

1.3 Scientific Visualization

Will Schroeder, Ken Martin, and Bill Lorensen, *The Visualization Toolkit*, Prentice-Hall, 1998.

Greg Nielson, Hans Hagen, and Heinrich Müller, *Scientific Visualization: Overviews, Methodologies, Techniques*, IEEE Computer Society Press, 1997.

Lenny Lipton, *The CrystalEyes Handbook*, StereoGraphics Corporation, 1991.

Brand Fortner, *The Data Handbook: A Guide to Understanding the Organization and Visualization of Technical Data*, Spyglass, 1992.

William Kaufmann and Larry Smarr, *Supercomputing and the Transformation of Science*, Scientific American Library, 1993.

Robert Wolff and Larry Yaeger, *Visualization of Natural Phenomena*, Springer-Verlag, 1993.

David McAllister, *Stereo Computer Graphics and Other True 3D Technologies*, Princeton University Press, 1993.

Peter Keller and Mary Keller, *Visual Cues: Practical Data Visualization*, IEEE Press, 1993.

1.4 Color and Perception

Roy Hall, *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, 1989.

David Travis, *Effective Color Displays*, Academic Press, 1991.

L.G. Thorell and W.J. Smith, *Using Computer Color Effectively*, Prentice Hall, 1990.

Edward Tufte, *The Visual Display of Quantitative Information*, Graphics Press, 1983.

Edward Tufte, *Envisioning Information*, Graphics Press, 1990.

Edward Tufte, *Visual Explanations*, Graphics Press, 1997.

Howard Resnikoff, *The Illusion of Reality*, Springer-Verlag, 1989.

1.5 Rendering

Andrew Glassner, *An Introduction to Ray Tracing*, Academic Press, 1989.

Steve Upstill, *The RenderMan Companion*, Addison-Wesley, 1990.

Ken Joy et al, *Image Synthesis*, IEEE Computer Society Press, 1988.

1.6 Fractals

Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe, *Chaos and Fractals: New Frontiers of Science*, Springer-Verlag, 1992.

Heinz-Otto Peitgen and Dietmar Saupe, *The Science of Fractal Images*, Springer-Verlag, 1988.

Michael Barnsley, *Fractals Everywhere*, Academic Press, 1988.

Benoit Mandelbrot, *The Fractal Geometry of Nature*, W.H. Freeman, 1977.

Przemyslaw Prusinkiewicz and James Hanan, *Lindenmayer-Systems, Fractals, and Plants*, Lecture Notes in Biomathematics #79, Springer-Verlag, 1989.

Heinz-Otto Peitgen and P.H. Richter, *The Beauty of Fractals*, Springer-Verlag, 1986.

Przemyslaw Prusinkiewicz and Aristid Lindenmayer, *The Algorithmic Beauty of Plants*, Springer-Verlag, 1990.

1.7 Animation

Alan Watt and Mark Watt, *Advanced Animation and Rendering Techniques*, Addison-Wesley, 1992.

Nadia Magnenat Thalmann and Daniel Thalmann, *Interactive Computer Animation*, Prentice-Hall, 1996.

Philip Hayward and Tana Wollen, *Future Visions: New Technologies of the Screen*, Indiana University Press, 1993.

1.8 Virtual Reality

John Vince, *Virtual Reality Systems*, Addison-Wesley, 1995.

1.9 The Web

Andrea Ames, David Nadeau, John Moreland, *The VRML 2.0 Sourcebook*, John Wiley & Sons, 1997.

Bruce Eckel, *Thinking in Java*, Prentice-Hall, 1998.

David Flanagan, *Java in a Nutshell*, O'Reilly & Associates, 1996.

David Flanagan, *Java Examples in a Nutshell*, O'Reilly & Associates, 1997.

Henry Sowizral, Kevin Rushforth, and Michael Deering, *The Java 3D API Specification*, Addison-Wesley, 1998.

1.10 Miscellaneous

OpenGL 1.1 Reference Manual, Addison-Wesley, 1997.

OpenGL 1.1 Programming Guide, Addison-Wesley, 1997.

Anne Spalter, *The Computer in the Visual Arts*, Addison-Wesley, 1999.

Ben Shneiderman, *Designing the User Interface*, Addison-Wesley, 1997.

Clark Dodsworth, *Digital Illusion*, Addison-Wesley, 1997.

Isaac Victor Kerlow and Judson Rosebush, *Computer Graphics for Designers and Artists*, Van Nostrand Reinhold, 1986.

William Press, Saul Teukolsky, William Vetterling, and Brian Flannery, *Numerical Recipes in C*, Second Edition, Cambridge University Press, 1997.

2. Periodicals

Computer Graphics Quarterly: published by ACM SIGGRAPH (www.siggraph.org, 212-869-7440)

Computer Graphics and Applications: published by IEEE (computer.org, 714-821-8380)

Transactions on Visualization and Computer Graphics: published by IEEE (computer.org, 714-821-8380)

Computer Graphics World: published by Pennwell (603-891-0123)

Journal of Graphics Tools: published by A.K. Peters (www.akpeters.com, 617-235-2210)

Transactions on Graphics: published by ACM (212-869-7440)

Computers in Science and Engineering (used to be *Computers in Physics*): published by the American Institute of Physics

IRIS Universe, newsletter from Silicon Graphics

Cray Channels, newsletter from Cray Research

3. Professional organizations

ACM..... Association for Computing Machinery
www.acm.org, 212-869-7440

SIGGRAPH... ACM Special Interest Group on Computer Graphics
www.siggraph.org, 212-869-7440

SIGCHI..... ACM Special Interest Group on Computer-Human Interfaces
www.acm.org/sigchi, 212-869-7440

IEEE Institute of Electrical and Electronic Engineers
computer.org, 202-371-0101

NAB National Association of Broadcasters
www.nab.org, 800-521-8624

ASME..... American Society of Mechanical Engineers
www.asme.org, 800-THE-ASME

4. Visual Stuff You Can Buy

SIGGRAPH Proceedings, Slides, and Video Reviews:

ACM Order Department
PO Box 12114
Church Street Station
New York, NY 10257
800-626-6626
FAX: 212-944-1318
<http://www.acm.org/catalog/>

5. Conferences

ACM SIGGRAPH:

1999: Los Angeles, CA – August 8-13
www.siggraph.org/s99
2000: New Orleans
2001: Los Angeles

IEEE Visualization:

1999: San Jose, CA – October 24-29
www.erc.msstate.edu/conferences/vis99
2000: Rutgers, NJ
2001: San Diego

NAB:

2000: Las Vegas, NV – April 10-13
www.nab.org

ACM SIGCHI:

2000: The Hague, Netherlands – April 1-6
www.acm.org/sigchi

ACM SIGARCH / IEEE Supercomputing:

1999: Portland, OR – November 13-19
www.supercomp.org

6. Graphics Performance Characterization

The GPC web site tabulates graphics display speeds for a variety of vendors' workstation products. To get the information, ping:

<http://www.specbench.org/gpc>

Glossary of Introductory Computer Graphics Terms

This glossary was taken from the Glindex chapter (combined glossary and index) of the introductory book *The Way Computer Graphics Works*, by Olin Lathrop, published by John Wiley and Sons, 1997, ISBN 0-471-13040-0.

Copyright Notice

This document is copyright 1997 by Olin Lathrop. No part of this document may be re-published, re-transmitted, saved to disk, or otherwise copied except as part of the necessary and normal operation of the software used to view this document.

An on-line version of this glossary is available at <http://www.cognivis.com/book/glossary.htm>.

Click on any of the following letters to jump to the section for that letter.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

2D

Two dimensional.

2D display controller

A type of display controller that is intended for 2D operations, like windows, popup menus, text, etc. This kind of display controller can usually not interpolate color values within a primitive, and therefore lacks support for 3D drawing.

2 1/2 D display controller

A 2D display controller that is able to perform the 2D operations required to display 3D primitives. This usually means color interpolation and Z buffering.

3D

Three dimensional.

3D display controller

A type of display controller that fully supports 3D operations and primitives. At a minimum, such a display controller can accept a 3D triangle, apply a 3D transform, perform the apparent color determination at the vertices, project it onto the 2D bitmap, and draw it with hidden surfaces

suppressed. Today's 3D display controllers use the Z buffer algorithm for hidden surface suppression.

3D texture

A texture map that is a function of three variables. This is also called a "solid texture", because the texture map is a volume. Solid textures have been used in diffuse color texture mapping to simulate things like marble and wood grain. Sometimes specifying the color in a volume is simpler than specifying it on the surface of an object, especially if the object has a complex shape.

3D transform

The act of converting the coordinates of a point, vector, etc., from one coordinate space to another. The term can also refer collectively to the numbers used to describe the mapping of one space on another, which are also called a "3D transformation matrix."

3D transformation matrix

See "3D transform."

- - - - **A** - - - -

adaptive space subdivision

Space subdivision refers to the process of breaking up the scene space into many small regions. Adaptive means this is only done where and when needed, instead of in a fixed way up front. Adaptive space subdivision is often used by ray tracers. Octrees and BSP trees are examples of subdivision algorithms that can be adaptive.

addition, vector

See "vector addition."

aliasing

The phenomenon that makes smooth lines and edges appear stair stepped or jagged. Aliasing is also called the "jaggies".

aliasing, temporal

See "temporal aliasing."

alpha buffer

The collective name for the alpha values for every pixel of an image or bitmap.

alpha buffered rendering

Using an alpha buffer for rendering, as opposed to for image compositing or matting. Alpha buffered rendering implies the ability to render semi-transparent primitives.

alpha buffering

The process of rendering or compositing images using an alpha buffer. An alpha buffer supplies an opacity fraction for every pixel.

alpha value

The alpha buffer value for a single pixel. An alpha value is a value indicating the pixels opacity. Zero usually represents totally transparent (invisible) and the maximum value represents completely opaque. Alpha values are commonly represented in 8 bits, in which case transparent to opaque ranges from 0 to 255.

ambient light

A light source that shines equally on everything. This is a hack used to give some illumination to areas that are not in direct view of any light source. In the real world, such areas are illuminated indirectly by light bouncing off other objects. Ambient illumination is commonly used except in radiosity rendering, because radiosity actually computes light bouncing between objects.

AND operator

A constructive solid geometry (CSG) modeling operation on two objects. The resulting object exists only where both input objects existed. This operation is also call INTERSECTION.

animation

Any method that can make an image appear to change over time. In computer graphics, this is done by showing many still images in rapid succession. This produces the illusion of a moving, or animated, image.

animation, inverse dynamics

See "inverse dynamics."

animation, inverse kinematics

See "inverse kinematics."

animation, key frame

See "key frame animation."

animation, parametric

See "parametric animation."

anti-aliasing

The process of reducing aliasing, or jaggies, in creating an image.

anti-aliasing filter, box

A box filter used in anti-aliasing averages all the samples of a high resolution image within each resulting pixel. All the samples are weighted equally over a rectangular region, usually the resulting anti-aliased pixel. Box filtering provides fair to medium quality results, but is much less complex than higher quality methods.

anti-aliasing filter, good quality

A good quality anti-aliasing filter blends values from a high resolution image such that samples near the resulting pixel center are weighted more than others. Sample weights smoothly approach zero at a distance of about $1\frac{1}{4}$ pixels.

anti-aliasing filter, simple

See "anti-aliasing filter, box."

apparent color determination

See "color determination."

apparent surface orientation

The orientation (which direction it's facing) a surface appears to have in an image. This is controlled by the shading normal vector, which is not necessarily the same as the normal vector of the primitive as it's actually drawn (the geometric normal vector).

- - - - **B** - - - -

B spline

A particular type of spline, the mathematical details of which are beyond the scope of this book.

back end

See "video back end".

basis vector

A vector that defines one axis of a coordinate system. Three basis vectors, one each for the X, Y and Z axis are needed to define a 3D coordinate system. A basis vector indicates the length and direction of a +1 increment in its axis.

beam current

The current of an electron beam in a cathode ray tube. The current is the number of electrons per unit time, which is usually measured in milliamperes. A higher beam current produces a brighter phosphor dot.

beta spline

A particular type of spline, the mathematical details of which are beyond the scope of this book.

bi-cubic patch

A particular type of surface patch. Bi-cubic surface patches can have curved edges, as opposed to polygons, which always have straight edges. The mathematical details of bi-cubic patches are beyond the scope of this book.

bi-quadratic patch

A particular type of surface patch. Bi-quadratic surface patches can have curved edges, as opposed to polygons, which always have straight edges. The mathematical details of bi-quadratic patches are beyond the scope of this book.

bi-linear interpolation

Interpolation is the process of determining plausible in-between values, given explicit values at particular points. Linear means that the values fall along a line from one known point to the next. This means the value changes a fixed amount for a fixed-sized step. Bi-linear means this process is carried out in two dimensions. In computer graphics, bi-linear interpolation is often applied to find color values at the interior pixels of a primitive. The apparent color values are computed explicitly at the vertices of a polygon and are bi-linearly interpolated in the polygon's interior. Bi-linear interpolation of pixel color values is also called Gouraud shading.

binary space partition

See "BSP tree."

bitmap

The collective name for all the stored pixels in a display controller. The bitmap is also the interface between the display controller's drawing front end and its video back end. The term bitmap is also used in general to refer to any 2D array of pixels.

blobbies

A name sometimes applied to potential functions used in modeling objects.

blue screen

A background often used for photographs or video that are to be matted, or composited, over other images. A blue background is almost universally used in chroma keying video compositing.

boolean operator

A mathematical operator that works on true or false values. These are also called logical operators. In computer graphics, constructive solid geometry (CSG) operators may also be called boolean operators. Some common CSG operators are AND, OR, NOT, and XOR.

bounding volume

A volume that encloses multiple 3D primitives. If the bounding volume doesn't intersect an object of interest (such as a ray in ray tracing), then the objects within the bounding volume are guaranteed to also not intersect the object, eliminating the need to check explicitly.

box filter

See "anti-aliasing filter, box."

BSP tree

A hierarchical method of subdividing a volume. BSP stands for "binary space partition." In this method, the volume is originally represented as one whole. If more detail is needed, the volume is subdivided into two volumes. Each of these are further subdivided into two volumes if more detail is needed. The process is repeated until the desired level of detail is achieved, or an arbitrary subdivision limit is reached.

BSP tree, regular

A special form of BSP tree where all the volume elements are rectangular solids that are always subdivided exactly in half along one of their three major axes.

bump mapping

A form of texture mapping where the texture supplies small perturbations of the shading normal vector.

- - - - C - - - -

calligraphic

An early type of computer graphics display that could only draw lines, not filled in areas. Calligraphic displays are rarely used today, and have mostly been replaced by raster displays.

camera point

See "eye point."

cathode ray tube

A type of vacuum tube that is commonly used as a computer graphics output device. A thin beam of electrons is shot at a spot on the inside of the tube's face. The inside of the face is coated with phosphors that emit light when the beam hits them. The beam is swept in a raster pattern to hit every spot on the screen. The beam current is modulated to make light and dark areas on the phosphors, forming an image. Cathode ray tube is usually abbreviated as CRT.

chroma keying

An image compositing technique commonly used on video signals. An overlay video signal is selected instead of a background video signal whenever the overlay isn't a particular preset hue, usually blue. Action shot in front of a blue screen can thereby appear on top of the background signal.

circular reasoning

See "reasoning, circular."

color determination

The process of figuring out what the apparent color of a particular point on a particular primitive is. This process is used to answer the question "What color is the object at this pixel?"

color, diffuse

See "diffuse color."

color, emissive

See "emissive color."

color, specular

See "specular color."

color index value

The input value to a color lookup table (LUT) of a display controller's video back end operating in pseudo color mode. Color index values are also referred to as pseudo colors.

color lookup table

A table of color values in a display controller's video back end. In pseudo color mode, it translates the pseudo color values into RGB color values. In true color mode it becomes three separate tables, one for each red, green, and blue component. It then translates the red, green, and blue pixel component values to the final displayed red, green and blue component values. The color lookup table is usually just called the LUT.

color purity

The degree to which a color CRT can display just one of its three red, green, or blue primary colors without displaying any portion of the other two. This is a measure of how much each electron gun can only hit the phosphor dots of its color.

color space

A scheme for describing different shades or colors. The RGB color space defines a shade as a mixture of specific quantities of red, green, and blue.

color space, RGB

See "RGB."

color space, IHS

See "IHS."

color wheel

A circular diagram of colors. The hue varies as the angle within the disc. Saturation increases from the center outward. The entire disc is usually shown at the same intensity.

compositing

The process of combining two images to yield a resulting, or composite, image. Alpha buffering is a common compositing technique in computer graphics.

compression

As used in this book, the process of encoding an image that contains redundant information such that it requires less storage. Runlength and LZW encoding are examples of lossless compression techniques. JPEG and MPEG are examples of lossy compression techniques.

compression, JPEG

See "JPEG."

compression, lossless

See "lossless compression."

compression, lossy

See "lossy compression."

compression, LZW

See "LZW compression."

compression, MPEG

See "MPEG."

compression, runlength encoding

See "runlength encoding."

concave

A property of a polygon that has at least one vertex bulge inward instead of outward. See the text for a more rigorous definition.

constraint

A rule of an inverse kinematics or inverse dynamics animation system that must be adhered to in solving the motion of all the objects. For example, a constraint in animating a walking human might be "the lower end of the leg always remains joined to the foot at the ankle."

constructive solid geometry

A modeling technique where complex shapes are built by combinations of simple shapes. Shapes are combined with boolean operators such as AND, OR, XOR, MINUS, and NOT. For example, a box with a hole thru it could be defined as box minus cylinder.

control point

A point used in defining a spline.

convergence

The degree to which all three electron beams of a color CRT meet at the same spot on the screen. A poorly converged CRT will show the red, green, and blue components of the image slightly offset from each other. This makes the image look blurry and produces color fringes around the edges of objects.

convex

A property of a polygon that bulges outward at all vertices. See the text for a more rigorous definition.

convolution

A mathematical operation that applies a weighted average defined by one function onto another

function. This is a very loose definition. A rigorous detailed one is beyond the scope of this book. Anti-aliasing is often done with convolutions. coordinate space Error! Bookmark not defined. A reference frame that defines numerical values, or coordinates, over an area (2D) or volume (3D).

cross product

A mathematical operation performed on two vectors, yielding a third vector. The third vector is always at right angles to the first two.

CRT

See "cathode ray tube."

CSG

See "constructive solid geometry."

CSG operator

See "boolean operator."

CSG AND operator

See "AND operator."

CSG EXCLUSIVE OR operator

See "XOR operator."

CSG INTERSECTION operator

See "AND operator."

CSG MINUS operator

See "MINUS operator."

CSG NOT operator

See "NOT operator."

CSG OR operator

See "OR operator."

CSG UNION operator

See "OR operator."

CSG XOR operator

See "XOR operator."

curved patch

A primitive used to model a small piece, or patch, of a surface. A curved patch, as opposed to a polygon, can have edges that are not straight.

- - - - **D** - - - -

de-Gauss

An action performed on a CRT monitor to de-magnetize the CRT and any material near it. CRTs are very sensitive to external magnetic fields. Such fields can cause alignment, convergence, purity, and image distortion problems. Most CRT monitors automatically perform de-Gaussing when they are first switched on.

deflection yoke

Coils mounted on a CRT that are used to steer the electron beam.

depth buffer

See "Z buffer."

depth buffer rendering

See "Z buffer rendering."

depth value

See "Z value."

diffuse color

An object surface property defined in the Phong lighting model. An object's diffuse color is reflected equally in all directions. Visually, this is the not-shiny or dull color.

diffuse reflection

The light reflected from an object that is reflected equally in all directions. These are the kind of inter-object reflections that are modeled well by radiosity.

direct evaluation (of object color)

The process of determining the apparent color of an object at a particular point by direct evaluation

of the lighting model, as opposed to interpolating from previously determined values.

directional light

A light source that shines from the same direction at every point in the scene. This is a handy shortcut for modeling far away light sources.

displacement

A distance in a particular direction. A vector exactly describes a displacement.

displacement vector

One of the vectors that defines a new coordinate system in terms of an old. The displacement vector is the vector from the old coordinate system's origin to the new coordinate system's origin.

display adapter

See "display controller."

display controller

A piece of computer hardware that receives drawing commands from the processor and drives the display. Some display controllers are commonly called the "video card", "display adapter", "graphics card", or something similar.

display controller, 2D

See "2D display controller."

display controller, 2 1/2 D

See "2 1/2 D display controller."

display controller, 3D

See "3D display controller."

display controller, GUI engine

See "2D display controller."

dither pattern

The particular pattern of threshold values used in dithering. Some dither patterns are random, while others are applied as repeating tiles across the whole image.

dithering

A technique for increasing an image's apparent color (or gray scale) resolution without increasing the number of color (or gray) levels actually used, at the cost of adding a "grainy" look to the image.

dot pitch

A measure of how closely spaced the phosphor triads are on the face of a color CRT. The triads are arranged in a hexagonal pattern, and the dot pitch is the distance from the center of one triad to the center of any of its six neighbors. Dot pitch usually specified in millimeters. Typical values are from .2 to .3 mm.

dot product

A mathematical operation of two vectors that produces a scalar. If both vectors have a length of one (unit vectors), then the dot product is the perpendicular projection of one vector onto the other.

dot width

The width of a point, or dot, primitive. Unlike mathematical points, computer graphics point primitives must have finite width to be visible. Many graphics subsystems allow the application to specify a width for such point primitives.

DPI

Dots per inch. This is a common measure of how close individual dots are on some output devices, such as inkjet printers.

DRAM

Dynamic random access memory. Most computer main memories are implemented with DRAM.

drawing front end

See "front end."

dye sublimation printer

See "printer, dye sublimation."

- - - - **E** - - - -

electron gun

The part of a cathode ray tube (CRT) that emits the electron beam.

emissive color

An object surface property sometimes used with the Phong lighting model. An object's emissive color is independent of any illumination. It therefore appears as if the object were emitting the color.

even field

See "field, even."

exclusive or

See "XOR operator."

explicit surface modeling

A class of modeling techniques that define objects by providing an explicit list of patches that cover their surfaces. These surface patches can be either polygons or any of a number of curved surface patch types. Other modeling techniques work on volumes, or only define an object's surface implicitly.

eye point

The point, or coordinate, a scene is being viewed from. This is also called the "eye point" or "camera point".

eye ray

A ray in ray tracing that originated at the eye point. All recursively generated rays have an eye ray as their original ancestor.

eye vector

A vector from anywhere in the scene to the eye point. Eye vectors are usually unitized before use. The eye vector is needed in computing the apparent color when the object's surface properties include specular reflection.

- - - - **F** - - - -

facet shading

A shading method where the shading normal vector is taken from the geometric normal of the surface actually drawn. This makes the surface patches visible, especially if they are planar.

field

Half of a complete video frame when interlacing is used. The two fields are referred to as the odd and the even. Each field contains only every other scan line.

field, even

The first of the two fields that make up a frame in interlaced video.

field, odd

The second of the two fields that make up a frame in interlaced video.

film recorder

An computer output device that can write images to film.

filter, anti-aliasing, box

See "anti-aliasing filter, box."

filter, anti-aliasing, good quality

See "anti-aliasing filter, good quality."

filter, box

See "anti-aliasing filter, box."

filter kernel

The function that defines the relative weight of a point depending on its position. The relative weight is used in computing a weighted average. This is actually a convolution operation, which is commonly used in anti-aliasing.

filtering

This is a broad word which can mean the removal of coffee grinds from the coffee. However, within the narrow usage of this book, a filtering operation is the same as a convolution operation (see "convolution"). Anti-aliasing is usually done by filtering.

flat projection

A method of projecting a 3D scene onto a 2D image such that the resulting object sizes are not dependent on their position. Flat projection can be useful when a constant scale is needed throughout an image, such as in some mechanical drawings.

flat shading

A shading method where each pixel of a primitive is drawn with the same color.

form factors

The name for the illumination coupling factors between polygons used in radiosity. Each form

factor indicates how much light from one polygon will reach another polygon.

fractal

Something that has infinite detail. You will always see more detail as you magnify a small portion of a fractal. Mandelbrot set functions are examples of 2D fractals.

frame

One complete video image. When interlacing is used, a frame is composed of two fields, each containing only half the scan lines.

front end

The part of a display controller that receives drawing commands from the processor and writes the primitives into the bitmap.

- - - - G - - - -

gaze direction

The direction the virtual camera is pointed in the scene description. The center of the image will display whatever is along the gaze direction from the eye point.

geometric normal vector

A normal vector of the primitives as they are actually drawn. This often differs from the normal vector of the surface that was being modeled. Facet shading results when the shading normal vector is taken from the geometric normal vector.

GIF

A file format for storing images. GIF stands for Graphics Interchange format, and is owned by CompuServe, Inc.

Gouraud shading

See "interpolation, bi-linear."

graftal

A modeling technique where complex shapes are defined as relatively simple, recursive procedures.

graphic primitive

An object that the graphics system is capable of drawing into the bitmap. Examples are lines, points, and some polygons.

graphics card

See "display controller."

GUI engine

See "2D display controller."

- - - - H - - - -**hard copy**

A copy of computer data that is directly readable by a human without using the computer. A printout is a good example of a hard copy.

HLS

Another name frequently used for the same thing as IHS. See "IHS."

homogeneous

Constant throughout.

horizontal refresh rate

See "refresh rate, horizontal."

HSV

Another name frequently used for the same thing as IHS. See "IHS."

HTML

HyperText Markup Language. The text formatting language used by documents on the world wide web.

- - - - I - - - -**IHS**

A color space where colors are defined by their intensity, hue, and saturation attributes. This is sometimes referred to as HSV, which stands for "hue, saturation, value."

image

A two dimensional array of pixels that together form a picture.

image file

A computer file that contains an image.

implicit surface modeling

A class of modeling techniques that define an object by its surface, but without explicitly providing primitives that make up the surface. Examples are potential functions and iso-surfaces.

ink jet printer

See "printer, ink jet."

internet

The name for the global network connecting many computers to each other.

interpolation

The mathematical process of determining plausible in-between values, given explicit values at particular points. Pixel values of polygons are often interpolated from values explicitly calculated at the vertices. Interpolation is usually much faster than explicitly calculating values.

inverse dynamics

A method for specifying motion in an animation. Linkages and other constraints are defined for the objects. A final state is then specified for some of the objects, and the computer calculates the motion of all the objects so that the final state is reached. Unlike in inverse kinematics, dynamic properties are taken into account, such as momentum, friction, energy loss in collisions, etc.

inverse kinematics

A method for specifying motion in an animation. Linkages and other constraints are defined for the objects. A final state is then specified for some of the objects, and the computer calculates the motion of all the objects so that the final state is reached.

iso-surface

An implicit surface that exists wherever a continuous scalar field in a volume is at a particular value (the iso-value).

interpolation, bi-linear

See "bi-linear interpolation."

INTERSECTION operator

See "AND operator."

----- J -----

jaggies

See "aliasing."

JAVA

A platform-independent way of defining procedures for use with world wide web documents.

JPEG

A lossy image file compression technique for still images.

----- K -----

key frame

A selected frame of an animation at which all the scene state is defined. In the key frame animation method, the scene state at key frames is interpolated to create the scene state at the in-between frames.

key frame animation

An animation control method that works by specifying the complete scene state at selected, or key, frames. The scene state for the remaining frames is interpolated from the state at the key frames.

key frame interpolation, linear

----- L -----

laser printer

See "printer, laser."

left vector

A vector sometimes used to define the virtual camera orientation in a scene description. This is more typically done with an up vector.

light, ambient

See "ambient light."

light attenuation

The property of light to get dimmer with distance from the light source. Real light intensity is

proportional to $1/R^2$, where R is the distance from the light source.

light, directional

See "directional light."

light, point

See "point light source."

light, point with $1/R^2$ falloff

See "point light source with $1/R^2$ falloff."

light ray

A ray in ray tracing that is launched from a point on an object towards a light source. The ray is used to determine whether the light source is illuminating the point. If the ray reaches the light source without hitting anything, then the light source is illuminating the point.

light source

A scene object that illuminates other objects.

light, spot

See "spot light source."

light vector

A vector from a point on an object towards a light source. Light vectors are usually unitized before use. A light vector is needed for each light source in computing the apparent color when the object's surface properties include diffuse or specular reflection.

linear interpolation

Interpolation is the process of determining plausible in-between values, given explicit values at particular points. Linear means that the values fall along a line from one known point to the next. This means the value changes a fixed amount for a fixed-sized step. Sometimes the term "linear interpolation" is used to refer to "bi-linear interpolation".

linear shading

See "bi-linear interpolation."

logical operator

See "boolean operator."

lookat point

A point in the scene that will project to the center of the image. The gaze vector points from the eye point towards a lookat point.

lossless compression

A compression scheme (see "compression") where all data is preserved. The data may be compressed and de-compressed any number of times without causing any changes. The compression ratio of lossless compression schemes is generally lower than that of lossy schemes. Runlength and LZW encoding are examples of lossless compression schemes.

lossy compression

A compression scheme (see "compression") where some data may be irreversibly lost, in favor of a high compression ratio. Many lossy schemes can trade off between amount of loss and the compression ratio. JPEG and MPEG are examples of lossy compression schemes for images.

lookup table

See "color lookup table."

LUT

See "color lookup table."

LZW compression

A digital data compression scheme that works by identifying and compressing recurring patterns in the data. LZW stands for Lempel-Ziv and Welch. Unisys corporation now claims that LZW compression is covered by its U.S. patent number 4,558,302.

- - - - **M** - - - -

mach bands

An optical illusion caused by a sudden change in the rate of change of the brightness (discontinuities in the brightness' second derivative). This can give the appearance of a light or dark line at the sudden change.

mandelbrot set

A popular mathematical function that exhibits fractal characteristics.

material properties

See "surface properties."

MINUS operator

A constructive solid geometry (CSG) modeling operation on two objects. For the operation "a MINUS b", the resulting object exists wherever A existed without being coincident with B. This operation can also be expressed "a AND (NOT b)."

modeling

As used in this book, the process of creating a description of an object or scene for the purpose of subsequent rendering.

modeling, explicit surface

See "explicit surface modeling."

modeling, implicit surface

See "implicit surface modeling"

modeling, level of detail

modeling, polygon

See "polygon modeling."

modeling, potential functions

See "potential function."

modeling, procedural

See "procedural modeling."

modeling, space subdivision

See "space subdivision modeling."

monitor

A piece of computer hardware for the live display of output. A monitor is different from a CRT, in that a monitor is the complete user-accessible unit that includes a case, power supply, knobs, etc. Many monitors use a CRT as the main imaging component, but other technologies are also used, such as flat panels.

MPEG

A lossy image file compression technique for motion pictures or animation sequences.

-----N-----

normal vector

A vector pointing straight out from (at right angles to) a surface.

normal vector, geometric

See "geometric normal vector."

NOT operator

A constructive solid geometry (CSG) modeling operation on one object. The resulting object exists only where the original object did not.

NURBS

Non Uniform Rational B-Splines. A particular type of surface spline for making curved surface patches in modeling complex shapes. These type of splines are supported by many computer aided design (CAD) systems.

-----O-----

object hierarchy ray tracing

See "ray tracing, object hierarchy."

octree

A hierarchical method for subdividing a volume. In this method, the volume is originally represented as one rectangular box (parallelepiped, to be more precise). If more detail is needed, the box is split exactly in half along each of its three major dimensions, yielding eight smaller boxes. This process is repeated on each sub-box until the desired level of detail is achieved, or an arbitrary subdivision limit is reached. Octrees are the 3D equivalent of quadrees.

odd field

See "field, odd."

OpenGL

A 3D graphics procedural interface. It was developed by Silicon Graphics, Inc., based on their earlier proprietary GL graphics library.

OR operator

A constructive solid geometry (CSG) modeling operator on two objects. The resulting object exists

where either or both input objects exist. This operation is also call UNION.

origin

The point in a coordinate space where all coordinates are zero.

orthographic projection

See "flat projection."

overlay where not zero

A simple image compositing method where the overlay image results wherever it's not zero, and the background image results wherever the overlay image is zero. This method is rather simplistic, but is sometimes supported in low-end hardware. It can be useful in overlaying text, for example. Alpha buffering is a more general compositing method.

- - - - P - - - -

palette image format

A format for storing an image that works much like pseudo color in a display controller. Each pixel contains a color ID instead of the actual color value. The true color represented by each color ID is defined in a table called the palette, which must also be stored with the image. A palette is much like a LUT in a pseudo color display controller.

parametric animation

An animation control method where scene state is determined by mathematical functions or computer procedures that take animation time as an input parameter.

particle system

A modeling technique where objects are defined by the collective tracks of many individual particles. Randomness is usually used to determine the details for each particle automatically, although overall guidance is supplied by the user.

persistence of vision

The property of the human visual system to continue seeing an image a short time (fraction of a second) after it has gone away.

perspective projection

A method of projecting a 3D scene onto a 2D image such that distant objects are displayed smaller than near ones. A normal camera produces images using perspective projection.

phong lighting model

A particular method for computing the apparent color of an object at a particular point.

phong shading

A shading method where the shading normal vector is interpolated for each pixel, then used in a separate apparent color calculation for that pixel.

phosphor (or phosphors)

The material coating the inside of a CRT face. Phosphors have the special property that they emit light when struck by an electron beam.

phosphor persistence

The property of phosphors to stay lit a short time (fraction of a second) after the electron beam is cut off or moved away.

phosphor triad

One red, green, and blue phosphor dot on the face of a color CRT. The dots are arranged in an equilateral triangle. Triads are arranged in a hexagonal pattern, such that each triad is the same distance from each of its six neighbors.

pixel

The smallest indivisible unit of a digital image. A pixel is always the same color throughout. An image is a two dimensional array of pixels.

point light source

A light source where all the light comes from one point. Although real light gets dimmer farther from a light source, a computer graphics point light source shouldn't be assumed to work that way unless explicitly stated.

point light source with $1/R^2$ falloff

A point light source where the light gets dimmer farther from the source. The light intensity is proportional to $1/R^2$, where R is the distance to the light source. This formula is used because that's how real light works.

point primitive

A graphic primitive that looks like a dot, or point.

polygon primitive

A graphic primitive that is an area enclosed by a loop of straight edges. Triangles and squares are

examples of polygon primitives.

polygon modeling

A modeling method where surfaces are approximated with abutting polygons.

polyline primitive

A graphic primitive of end-to-end line segments. A polyline primitive is more efficient than each of the line segments as separate vector primitives.

potential function

A type of implicit surface. See the text for details.

primitive

See "graphics primitive."

primitive, point

See "point primitive."

primitive, polygon

See "polygon primitive."

primitive, polyline

See "polyline primitive."

primitive, quad mesh

See "quad mesh primitive."

primitive, vector

See "vector primitive."

printer

Computer peripherals for producing hard copy on paper and other similar media.

printer, dye sublimation

A type of printer that works by evaporating controlled amounts of dye from a ribbon onto the page. The amount of dye can be accurately controlled, yielding continuous color resolution. Dye sublimation printers are relatively expensive, but produce output comparable in quality to the

traditional wet silver photographic process.

printer, ink jet

A type of printer that shoots tiny droplets of ink onto the page. Ink jet printers are a relatively low cost way to get computer graphics output.

printer, laser

A type of printer that works almost like a photocopier. The image is created by a laser under computer control, instead of coming from an original document as in a photocopier.

printer, thermal wax

See "printer, wax transfer."

printer, wax transfer

A type of printer that works by depositing small specs of wax from a ribbon onto the page. The ribbon is pressed against the page, and wax is transferred wherever the ribbon is heated. This type of printer is also called "thermal wax."

procedural model

An object model defined implicitly by a procedure that can produce volume or surface elements.

projection, flat

See "flat projection."

projection method

A scheme for mapping the 3D scene geometry onto the 2D image.

projection, orthographic

See "flat projection."

projection, perspective

See "perspective projection."

pseudo color system

A graphics system that stores pseudo colors, instead of true colors, in its bitmap. Pseudo colors are translated to true colors by the color lookup table (LUT).

pseudo color value

See "color index value."

- - - - Q - - - -

quad mesh primitive

A graphic primitive that contains a grid of quadrilaterals. A quad mesh primitive is more efficient than the equivalent separate quadrilateral primitives.

quadtree

A hierarchical method for subdividing an area. In this method, the area is originally represented as one box (parallelogram, to be precise). If more detail is needed, the box is split exactly in half along each of its two major dimensions, yielding four smaller boxes. This process is repeated on each sub-box until the desired level of detail is achieved, or an arbitrary subdivision limit is reached. Quadtrees are the 2D equivalent of octrees.

- - - - R - - - -

radiosity

A rendering method that takes into account diffuse reflection between objects.

raster scan

The name for the pattern the electron beam sweeps out on a CRT face. The image is made of closely spaced scan lines, or horizontal sweeps.

ray casting

A term sometimes used to denote non-recursive ray tracing.

ray, eye

See "eye ray."

ray, light

See "light ray."

ray tracing

A rendering method that follows rays of light backwards to eventually find what color they are. Rays may be launched recursively when the color of a point on an object depends on incoming light from specific known directions. For example, when a ray hits a reflective object, a recursive ray is launched in the direction the reflected light is coming from.

ray tracing, object hierarchy

A ray tracing speedup method where objects are kept track of in groups. These groups can be further grouped in a hierarchy. A bounding volume is maintained for each group in the hierarchy. If a ray doesn't intersect a bounding volume, then it definitely doesn't intersect any subordinate object in the hierarchy.

ray tracing, space subdivision

A ray tracing speedup method where the scene space is subdivided into blocks. A list is kept for each block indicating which objects a ray could hit in that block. As a ray is traced, it is walked thru the blocks, checking for intersection only with the objects listed in each block.

ray tracing speed issues

reasoning, circular

See "circular reasoning."

recursive ray tracing

See "ray tracing."

reflection vector

A vector used in the phong lighting model to compute the specular reflection. It is usually unitized, and points in the direction light is reflecting off the object.

refresh rate

The rate at which parts of the image on a CRT are re-painted, or refreshed. The horizontal refresh rate is the rate at which individual scan lines are drawn. The vertical refresh rate is the rate at which fields are drawn in interlaced mode, or whole frames are drawn in non-interlaced mode.

refresh rate, horizontal

The rate at which scan lines are drawn when the image on a CRT is re-drawn, or refreshed.

refresh rate, vertical

The rate at which fields are re-drawn on a CRT when in interlaced mode, or the rate at which the whole image is re-drawn when in non-interlaced mode.

regular BSP tree

See "BSP tree, regular."

rendering

The process of deriving an 2D image from the 3D scene description. This is basically the "drawing" step.

resolution

The measure of how closely spaced the pixels are in a displayed image. For example, if 1,024 pixels are displayed across a screen that is 12 inches wide, then the image has a resolution of 85 pixels per inch.

RGB

A color space where colors are defined as mixtures of the three additive primary colors red, green, and blue.

right vector

A vector sometimes used to define the virtual camera orientation in a scene description. This is more typically done with an up vector.

runlength encoding

A lossless digital data compression scheme. It identifies identical consecutive values, and replaces them with just one copy of the value and a repeat count.

- - - - S - - - -

scalar

A regular, single number, as opposed to a vector.

scan line

One line in a raster scan. Also used to mean one horizontal row of pixels.

scan line order

A way of arranging image data so that all the pixels for one scan line are stored or transmitted before the pixels for the next scan line.

scan rate

See "refresh rate."

scattered light

See "secondary scatter."

scene

The complete 3D description of everything needed to render an image. This includes all the object models, the light sources, and the viewing geometry.

secondary scatter

Light that is reflected from a non-emitting object that illuminates other objects. This is the kind of inter-object illumination that is computed by radiosity.

shading

This term is used several ways in computer graphics. However, shading always has something to do with figuring out pixel color values, as opposed to figuring out the geometry or which pixels are to be drawn.

shading, bi-linear

See "bi-linear interpolation."

shading, facet

See "facet shading."

shading, flat

See "flat shading."

shading, Gouraud

See "bi-linear interpolation."

shading, linear

See "bi-linear interpolation."

shading normal vector

The normal vector used in determining the apparent color of an object. In facet shading, the shading normal is the geometric normal of the primitives used to model the surface. In smooth shading, the shading normal is the normal vector of the surface that was modeled. The shading normal vector may be further modified by bump mapping.

shadow mask

A thin layer in a color CRT that the electron beams can not penetrate. It is suspended just in front of (from the electron beam's point of view) the phosphor screen. The shadow mask has one hole for each phosphor color triad. Due to the position of the phosphor triads, the shadow mask holes,

and the angle of the three electron beams, each electron beam can only hit the phosphor dots of its assigned color.

SIGGRAPH

The Special Interest Group on Graphics of the Association for Computing Machinery (ACM). SIGGRAPH is the premier professional association for computer graphics.

smooth shading

Any shading technique that attempts to make the rendered surface look as close as possible to what was modeled, instead of the primitives used to approximate that model. This is usually done by taking the shading normal vector from modeled surface, instead of from the geometric normal vector.

solid texture

See "3D texture."

space subdivision, adaptive

See "adaptive space subdivision."

space subdivision modeling

A modeling technique where the scene is broken into small regions of space. A list of the objects present is kept for each region. Examples of space subdivision modeling are BSP trees and octrees.

space subdivision ray tracing

See "ray tracing, space subdivision."

specular color

The color of an object's shiny highlights. Specular reflection also depends on the specular exponent. Both these parameters are part of the object's surface properties.

specular exponent

This is a scalar value that controls the "tightness" of the specular highlights. A value of zero causes the specular color to be reflected equally in all direction, just like the diffuse color. An infinite value causes it to be reflected the same way a mirror would. Common values are about 5 to 60.

spline, B

See "B spline."

spline, beta

See "beta spline."

spline patch

A curved surface which takes its shape from the placement of a set of control points.

spot light source

A light source that does not shine in all directions, usually in a cone. This is a convenient hack for modeling lamps with shades or reflectors, without actually having to compute the effect of the shade or reflector during rendering.

subpixel

An input pixel to an operation that uses multiple smaller pixels at higher resolution to compute each resulting pixel at the final resolution. This is done, for example, in an anti-aliasing filtering operation.

surface orientation, apparent

See "apparent surface orientation."

surface properties

The collective name for all the object-specific parameters that are used to compute the apparent color of a point on that object. These include the diffuse color, specular color, etc. The term "surface properties" is common but not standard. Other names for the same thing are "visual properties", or "material properties".

- - - - T - - - -

temporal aliasing

Aliasing in time by animation frame, instead of spatially by pixel. The visual effect of temporal aliasing has been called "strobing". Moving objects appear to jump thru a sequence of frozen steps, instead of in smooth paths.

tessellation level

The relative fineness or granularity of the patches used to model a surface. A model with smaller, and therefore more patches is said to have a higher tessellation level.

tessellation

The act of tiling a surface with individual surface patches.

texil

One pixel of a texture map, where the texture map is an image. In diffuse color texture mapping, the texture value is a color. The texture color values are therefore often supplied as an image. A texel is one pixel of this image, as opposed to one pixel of the final image being rendered.

texture, 3D

See "3D texture."

texture map

The external function that supplies the texture value in texture mapping. In diffuse color texture mapping, the texture map is a set of color values in two dimensions. This is usually specified as an image. Note, however, that not all texture maps are images, because not all textures are two dimensional color values. Bump mapping is a form of texture mapping where the texture is not an image, since the texture values are shading normal vector perturbations instead of colors.

texture mapping

The process where some parameter of the apparent color computation is replaced by an external function, called the texture map. A common example is texture mapping an object's diffuse color from an image. This gives the appearance of the image painted onto the object. Bump mapping is another form of texture mapping discussed in this book.

texture mapping, diffuse color

A form of texture mapping where the texture defines the object's diffuse color. This gives the appearance of pasting the texture image onto the object.

texture mapping, normal vector perturbations

See "bump mapping."

thermal wax printer

See "printer, wax transfer."

TIFF

Tag Image File Format. A common image file format. Just about all applications that can import image files support the TIFF format.

toner

The material that is deposited onto the page to form the image in a photocopier or laser printer.

transform, 3D

See "3D transform."

transformation matrix, 3D

See "3D transform."

transparency value

See "alpha value."

triad

See "phosphor triad."

triangle strip primitive

A graphic primitive that contains a set of successively abutting triangles. A triangle strip primitive is more efficient than the equivalent separate triangle primitives. It is also called a "Tstrip."

true color system

A graphics system that stores true colors, as opposed to pseudo colors, in its bitmap.

Tstrip primitive

See "triangle strip primitive."

----- **U** -----

UNION operator

See "OR operator."

unit vector

A vector of length, or magnitude, one.

unitize

To make a vector a unit vector. This means adjusting its length to be one without effecting the direction it's pointing.

up vector

A vector used to define the orientation of the virtual camera in a scene description.

----- **V** -----

vector

A value that has a direction and a magnitude (length). Plain numbers can be called "scalars" to explicitly distinguish them from vectors.

vector addition

The process of adding two vectors, which results in another vector.

vector, basis

See "basis vector."

vector cross product

See "cross product."

vector dot product

See "dot product."

vector, eye

See "eye vector."

vector, geometric normal

See "geometric normal vector."

vector, left

See "left vector."

vector, light

See "light vector."

vector primitive

A graphics primitive that looks like a line segment. Unlike mathematical line segments, vector primitives have finite widths.

vector, reflection

See "reflection vector."

vector, right

See "right vector."

vector scaling

An operation of a vector and a scalar. The resulting vector is the same as the input vector, except that its magnitude (length) is multiplied by the scalar. Scaling a vector by the reciprocal of its magnitude results in a unit vector.

vector, shading normal

See "shading normal vector."

vector thickness

The width of a vector, or line segment, primitive. Unlike mathematical line segments, computer graphics vector primitives must have a finite width to be visible. Many graphics systems allow the application to specify a width for such primitives.

vector, unit

See "unit vector."

vector unitize

See "unitize."

vector, up

See "up vector."

vertical refresh rate

See "refresh rate, vertical."

video back end

The part of a display controller that reads the pixel data in the bitmap and produces the live video signals. Among other components, the back end contains the color lookup tables (LUTs), the digital to analog converters (DACs), and the logic to generate the video timing signals.

video card

See "display controller."

view direction

See "gaze direction."

view point

See "eye point."

virtual reality

A name loosely applied to systems that attempt to immerse the user in a virtual world generated by the computer. This is often done with a stereoscopic display that is updated based on head position, and usually includes some sort of 3D pointing device. More advanced systems include 3D sound and some form of touch feedback. This is still a rapidly changing area.

visual properties

See "surface properties."

voxel

A unit of subdivided space. Voxel stands for "volume element". Nodes in an octree, for example, are referred to as voxels.

VRAM

Video random access memory. This is really DRAM with additional features specifically for use as bitmap memory in display controllers. VRAM typically costs twice as much as DRAM, but allows the drawing engine full access to the bitmap independent from the video back end. This can increase the hardware drawing rate.

VRML

Virtual Reality Modeling Language. The standard description language for 3D graphics in world wide web documents.

-----W-----

wax transfer printer

See "printer, wax transfer."

web browser

Software that can read and display world wide web documents.

wire frame

A rendering method where only the outline of objects and primitives are drawn.

world wide web

A set of interconnected documents on the internet that adhere to the HTML standard. The world wide web is often abbreviated to WWW.

WWW

See "world wide web."

- - - - **X** - - - -

XOR operator

A constructive solid geometry (CSG) modeling operation on two objects. The resulting object exists where each object existed alone, not coincident with each other. XOR is an abbreviation for "exclusive or."

X Windows

A window management and 2D graphics environment that is very common on Unix systems.

- - - - **Y** - - - -

- - - - **Z** - - - -

Z buffer

The collective name for all the Z values in a bitmap.

Z buffer rendering

A rendering method where an additional depth, or Z, value is kept per pixel. A pixel is only overwritten by a new primitive if the new Z value represents a shorter distance, or depth, from the eye point. The end result is an image of the front most surface of the front most objects.

Z value

The name for the value that represents distance from the eye point in a Z buffer rendering system.