Computer Graphics 2 - Object representations

Tom Thorne

Some slides are courtesy of Taku Komura
thomas.thorne@ed.ac.uk
www.inf.ed.ac.uk/teaching/courses/cg/

Last time

- History, applications
- Basic vector algebra
- Computer graphics pipeline

Overview

Polygon meshes

- Storage
- Representation
- Decomposition
- Quad meshes
- Generation
- Implicit surfaces

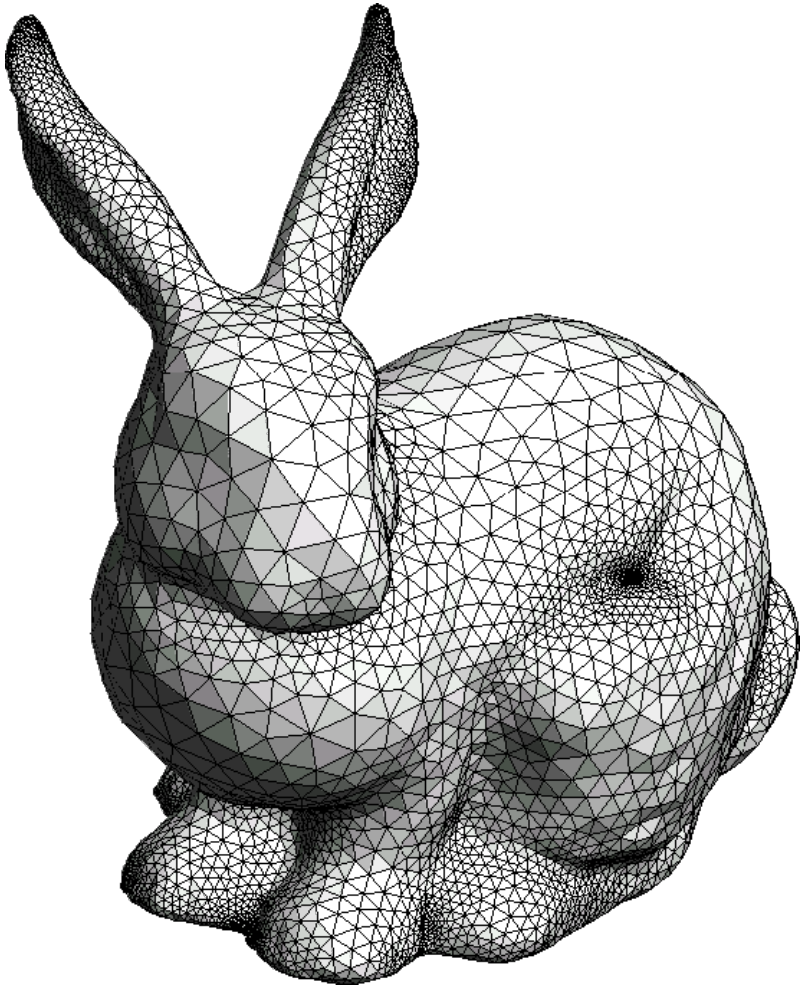
Level of detail

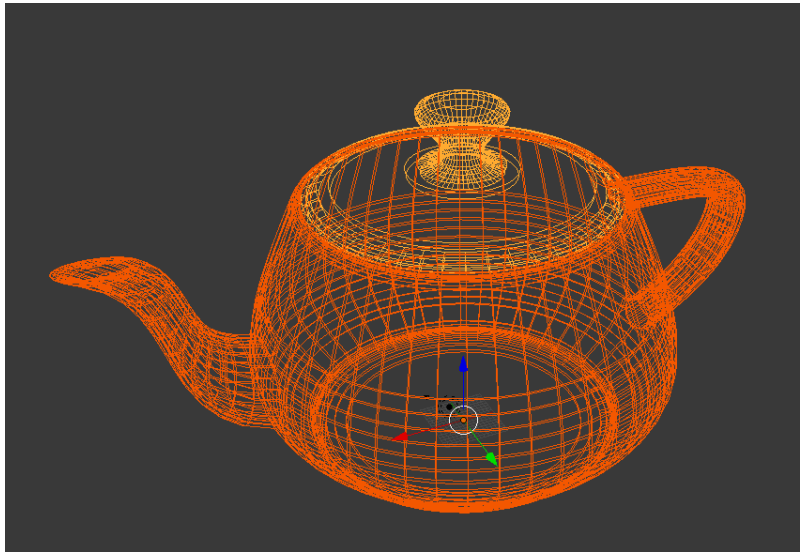
- Scaling methods

Image representations

- Pixel Buffers
- Alpha channels
- Z buffers

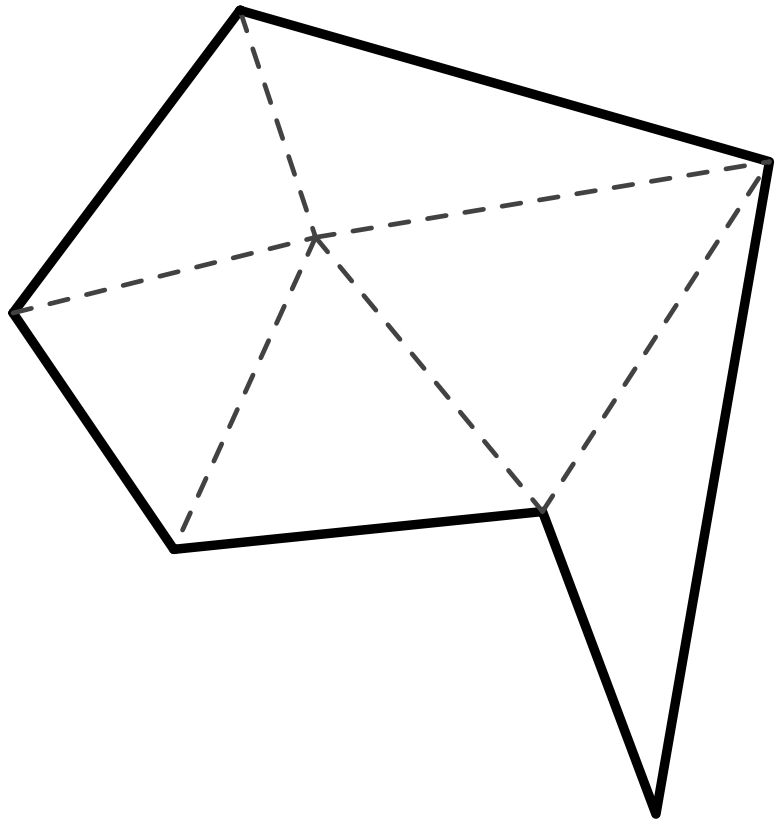
Mesh structures





Objects represented as a set of polygons

Triangle representation

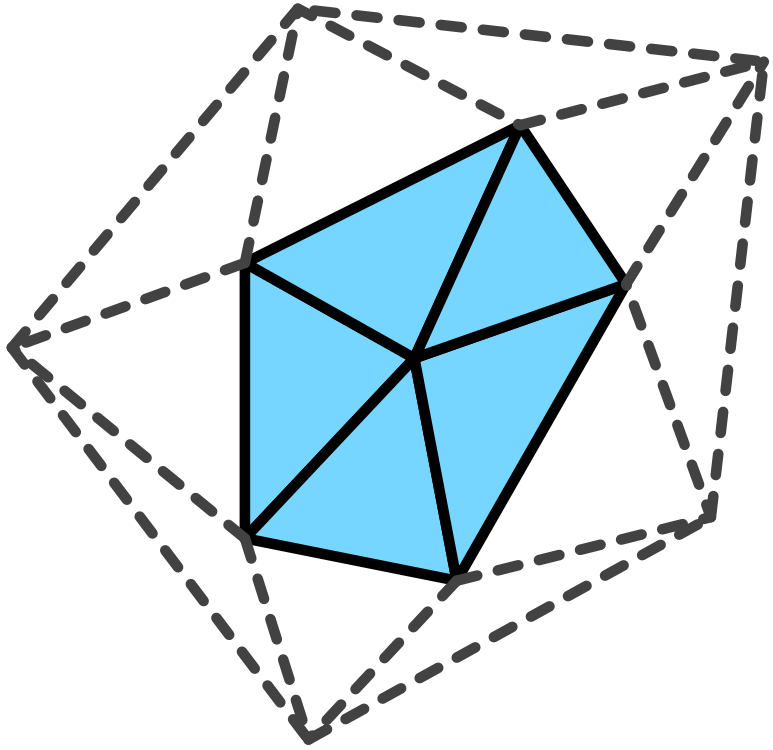


- Polygons can be broken down into triangles

Mesh topology

Manifolds:

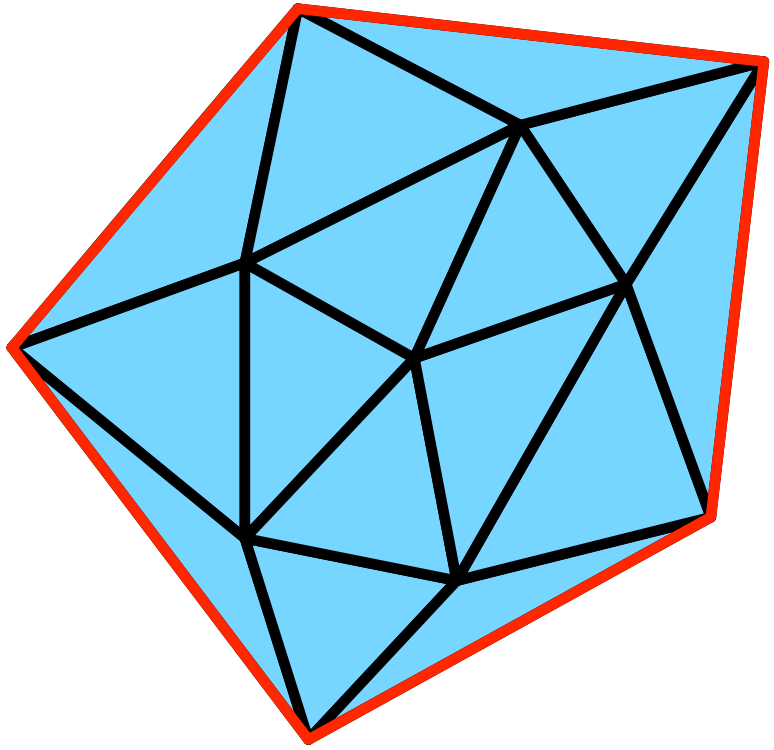
- All edges belong to two triangles
- All vertices have a single continuous set of triangles around them



Mesh topology

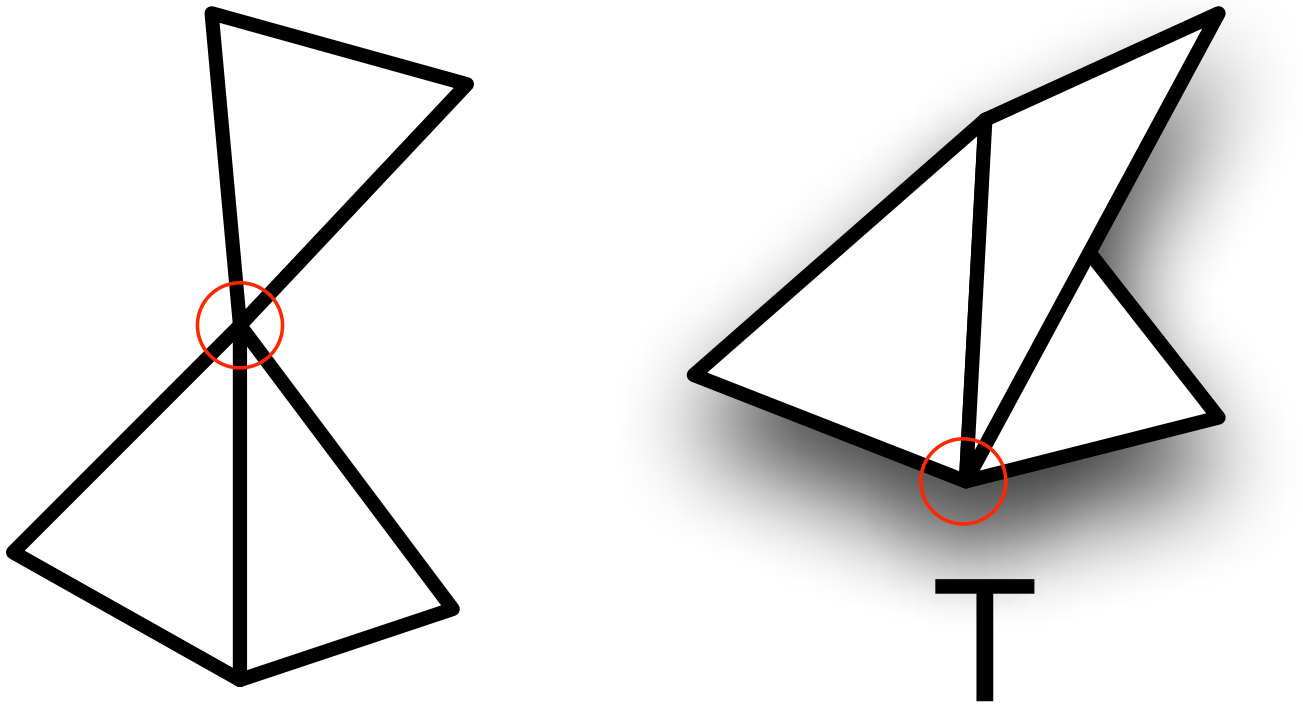
Manifolds with boundaries:

- All edges belong to one or two triangles
- All vertices have a single continuous set of triangles around them



Mesh topology

Examples of non-manifold meshes:



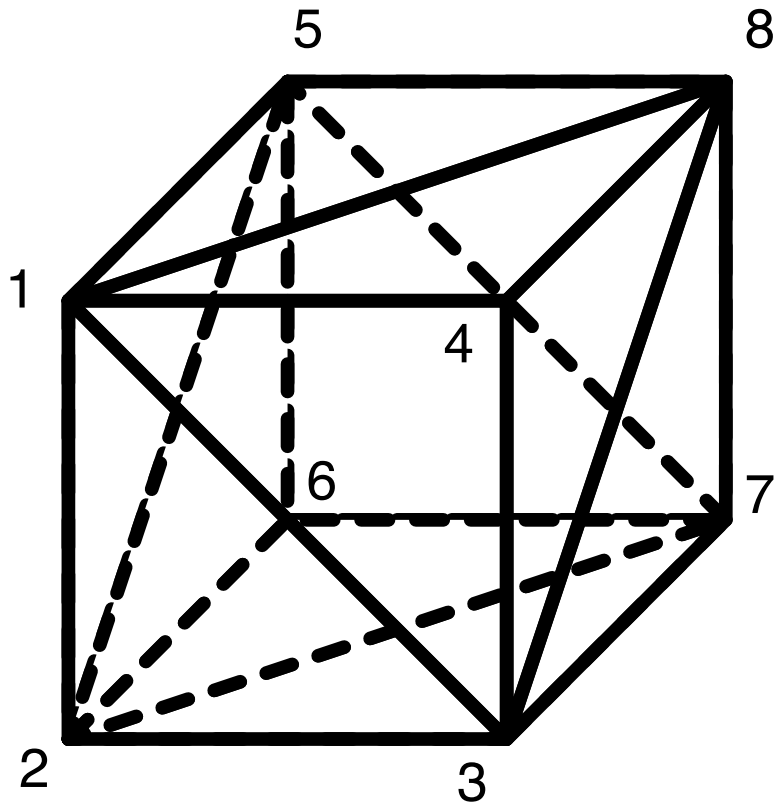
Storage format

Vertices:

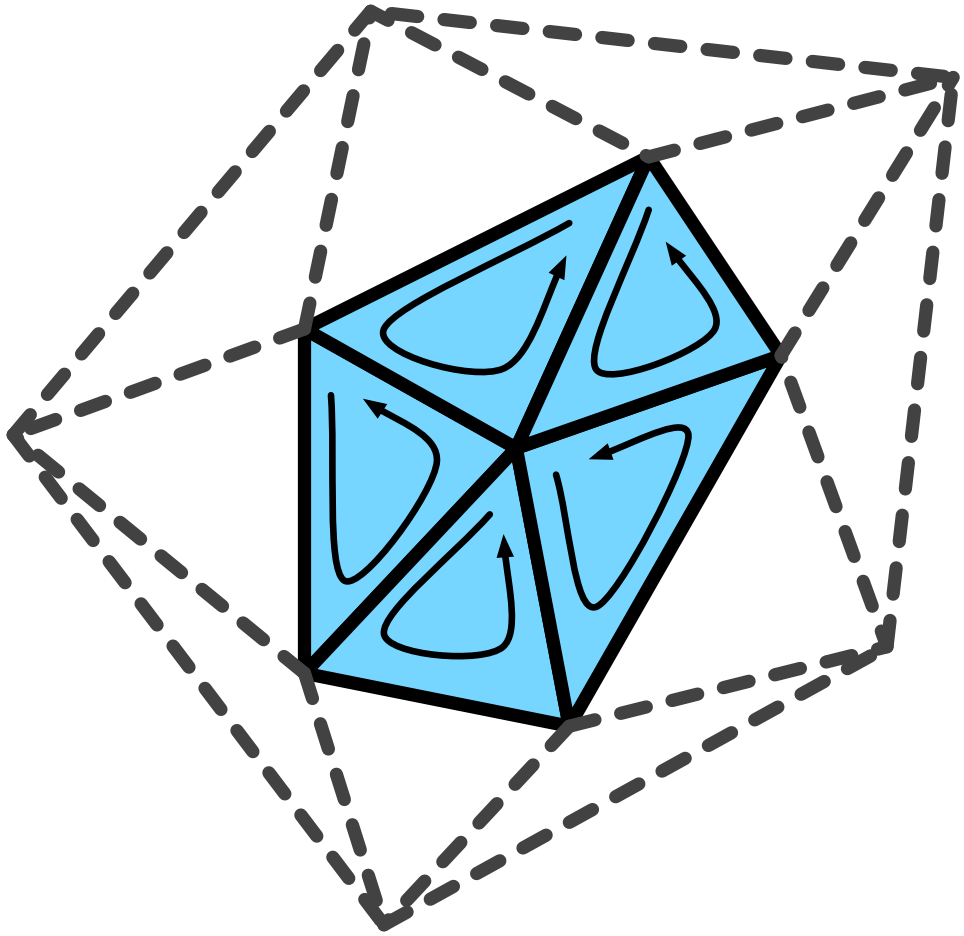
-1	1	1
-1	-1	1
1	-1	1
1	1	1
-1	1	-1
-1	-1	-1
1	-1	-1
1	1	-1

Polygons:

1	2	3
1	3	4
1	4	8
1	8	5
4	3	8
3	7	8
5	7	6
5	8	7
1	5	2
5	6	2
2	6	7
2	7	3

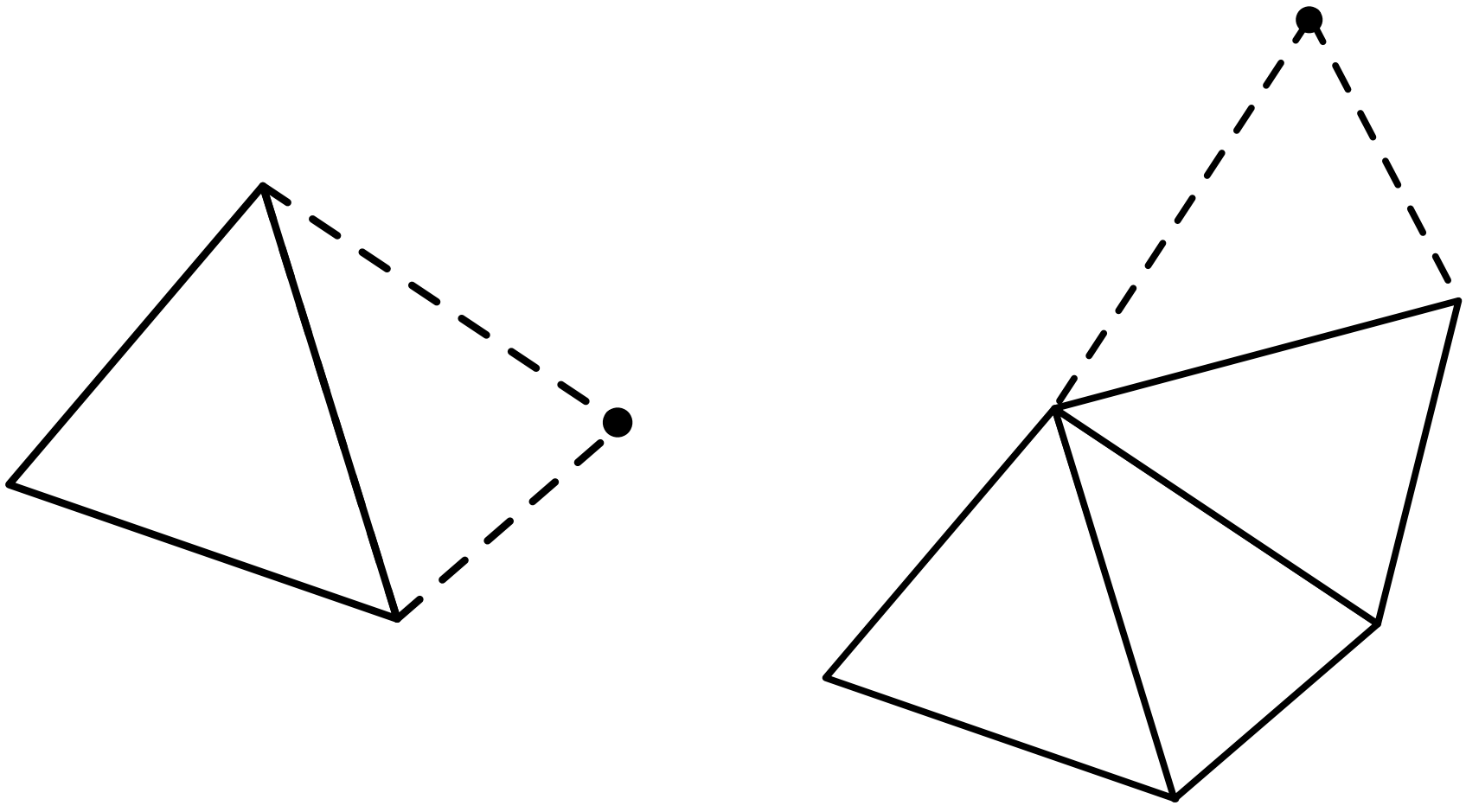


Orientation



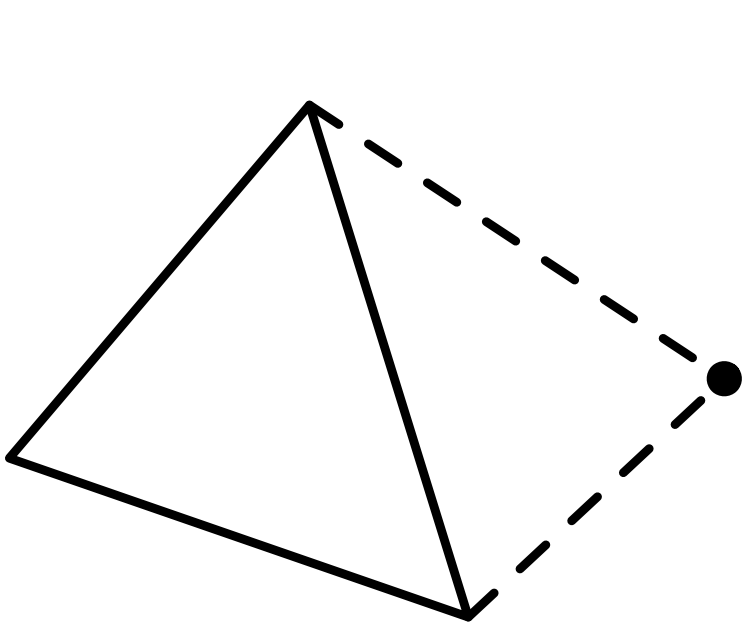
Vertexes in triangle list stored in counter clockwise order

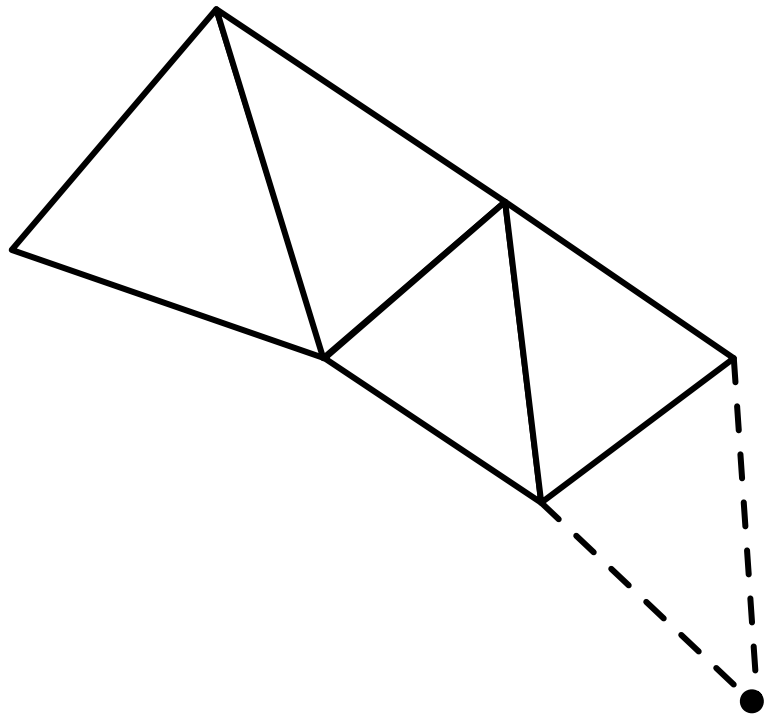
Triangle fans



Instead of storing $3T$ edges, store $T + 2$

Triangle strips





Instead of storing $3T$ edges, store $T + 2$

Decomposing surfaces

We can decompose meshes into triangle strips or fans to reduce storage requirements

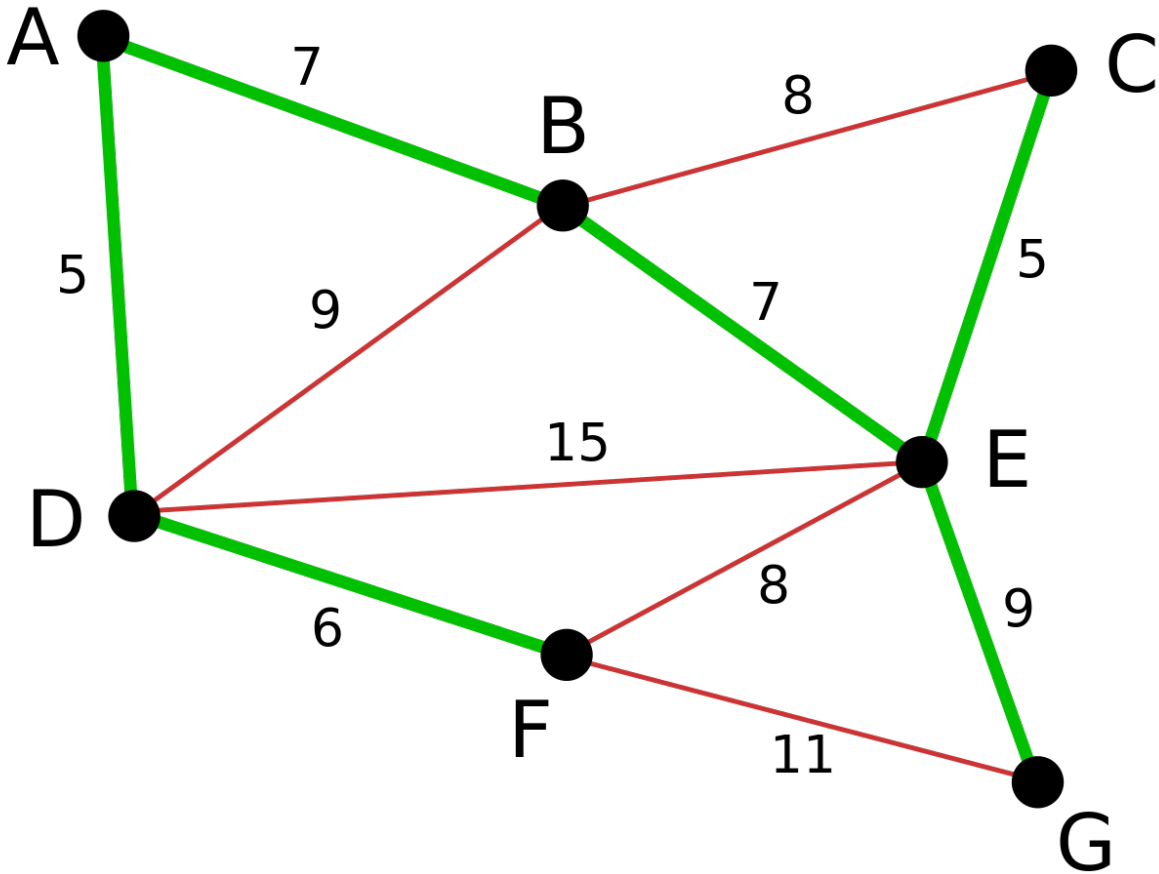
Some methods:

- Minimal spanning tree
- Sprials

Minimal spanning tree decomposition

A minimal spanning tree produces a tree over a graph which visits every node whilst minimising costs

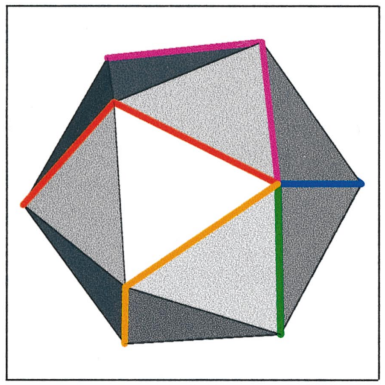
- edges are given costs
- a tree with minimal total cost is found

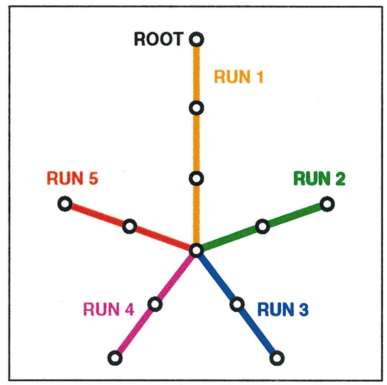


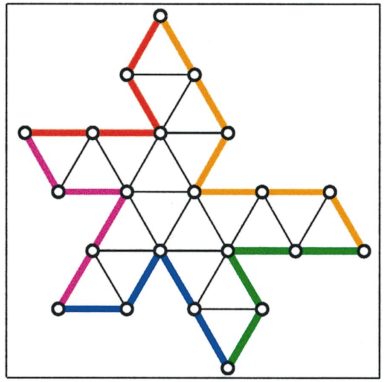
Minimal spanning tree decomposition

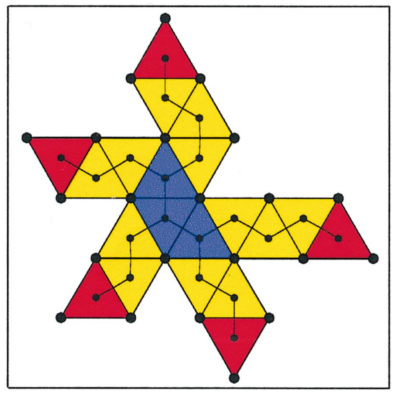
From a mesh, produce a graph where:

- nodes correspond to triangles of the mesh
- edges shared by a pair of triangles become edges in the graph
- cost is set to the Euclidean distance between the triangle and the root triangle



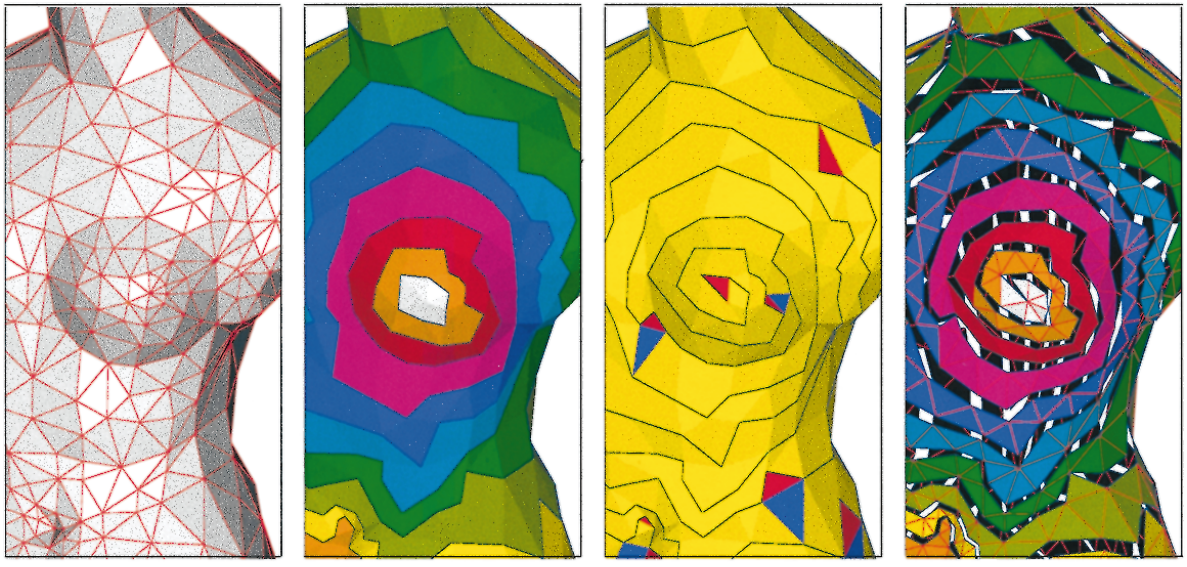






Sprial decomposition

- Generate layers of triangles
- Connect layers into a single spiral of triangles



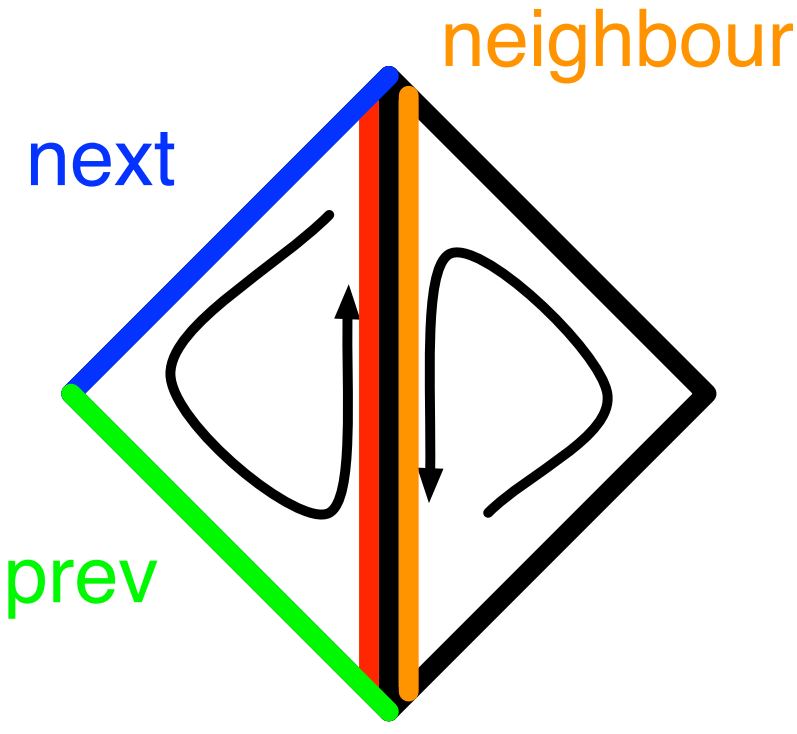
Directed edge data structure

Vertices

x_1	y_1	z_1	e_r
x_2	y_2	z_2	e_s
⋮	⋮	⋮	⋮

Edges

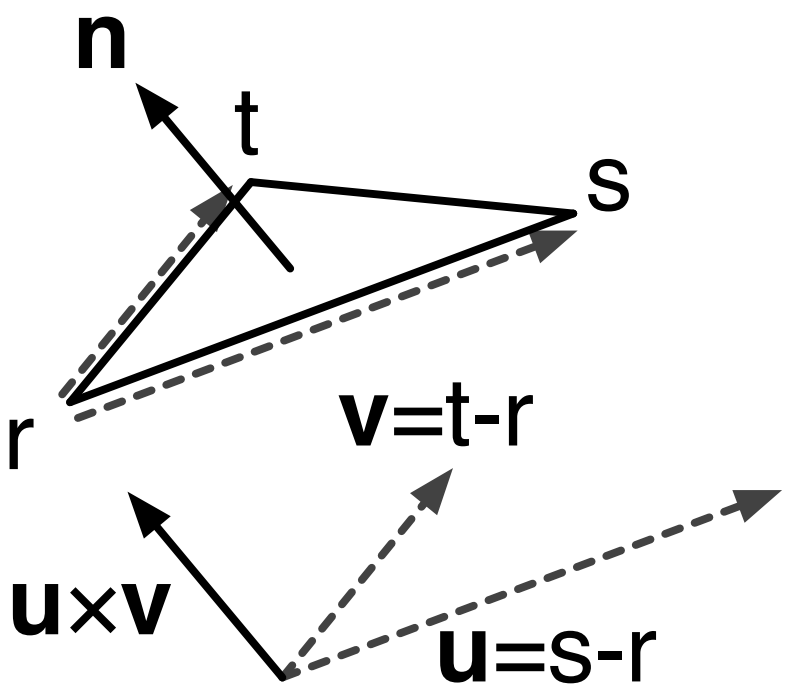
v_a	v_b	$e_{neighbour}$	e_{next}	e_{prev}
⋮	⋮	⋮	⋮	⋮

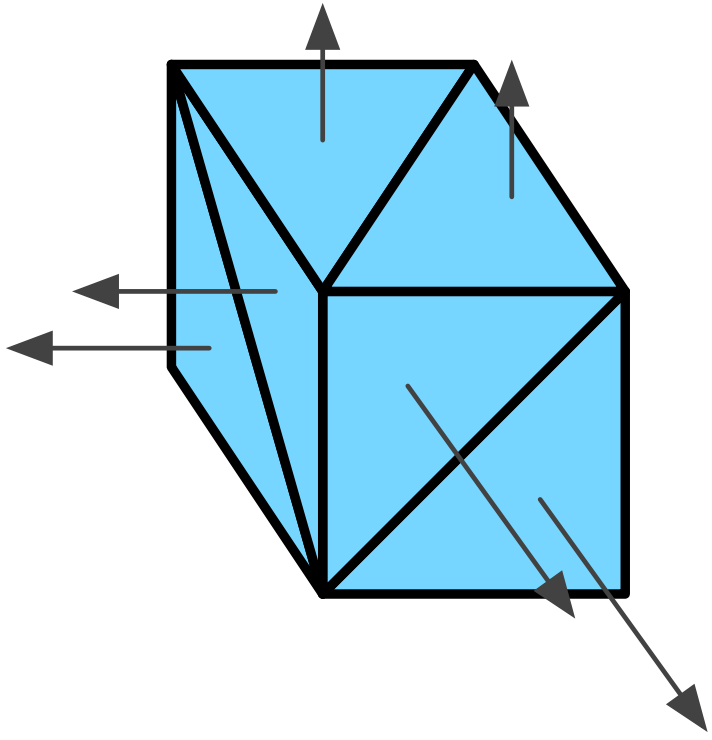


Surface normals

Calculation:

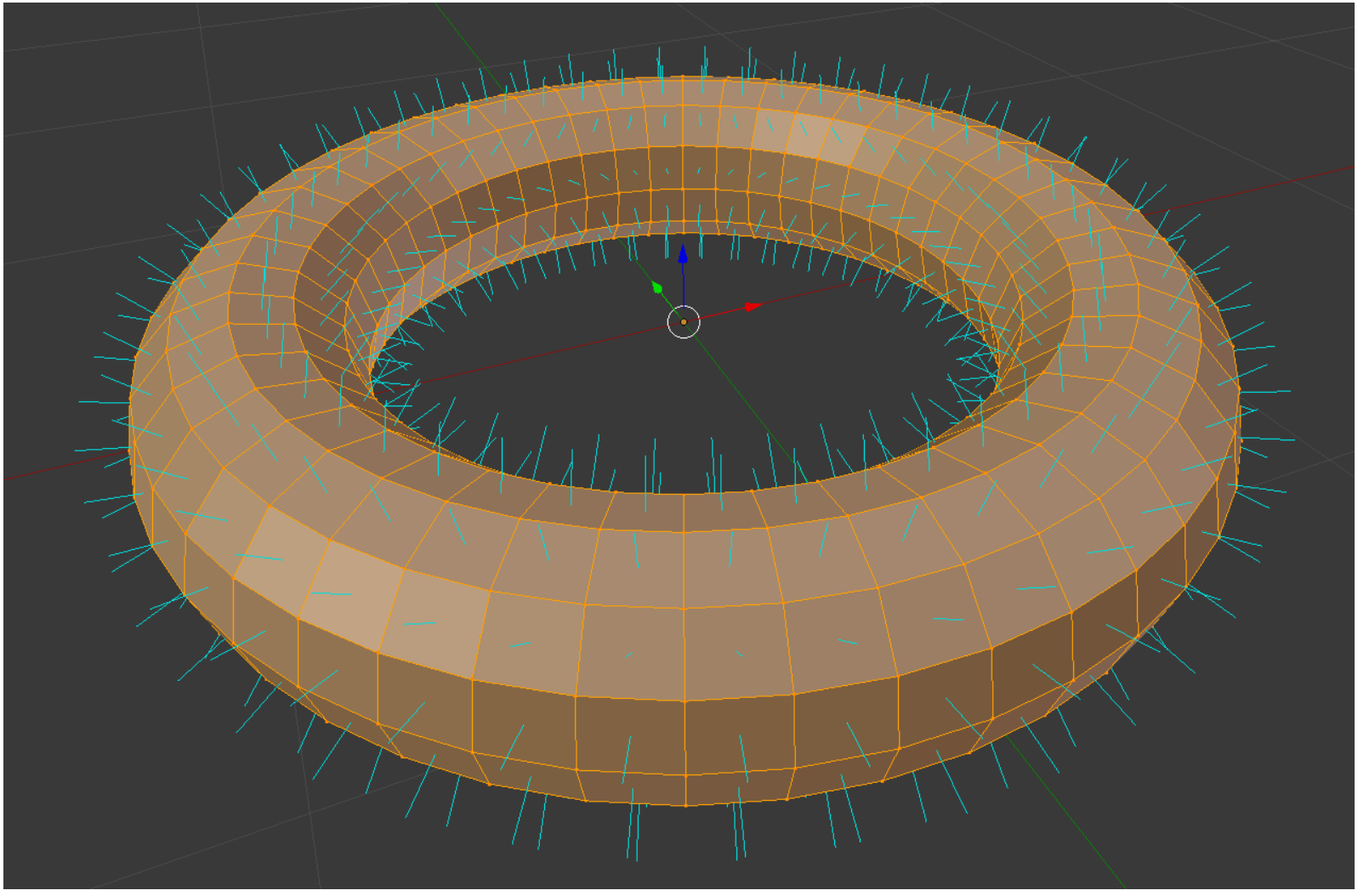
$$\boldsymbol{n} \quad = \quad (t - r) \times (s - r)$$





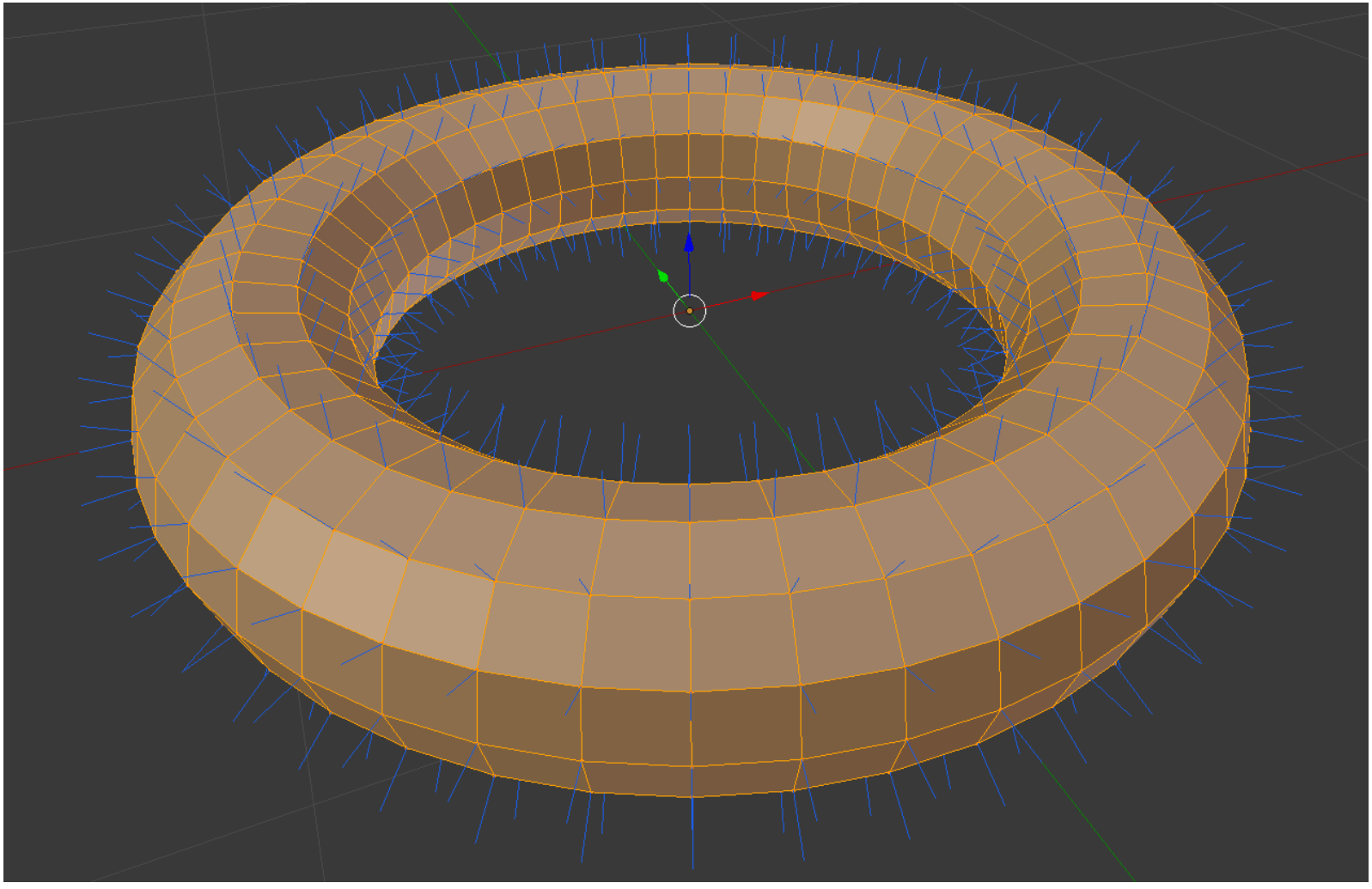
Surface normals

Face normals



Surface normals

Vertex normals



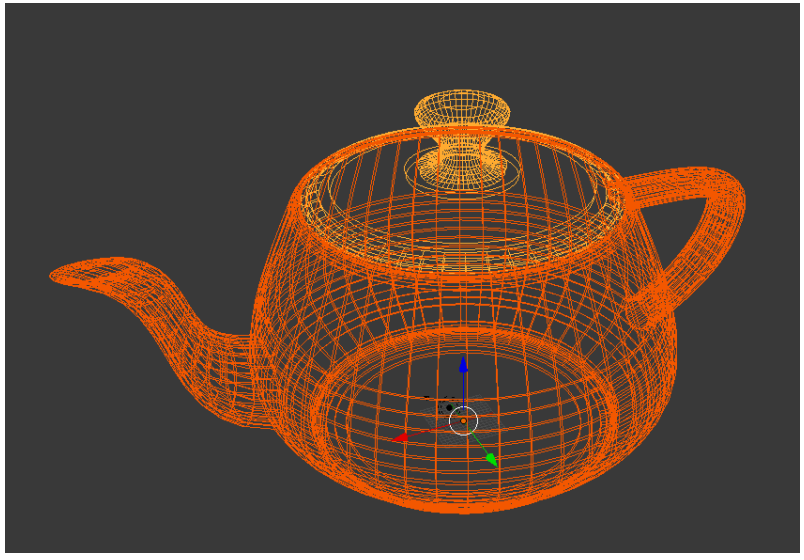
Quadrilateral meshes

Relatively easy to extend to quadrilateral meshes (quadmeshes):

- Same storage format

Problems:

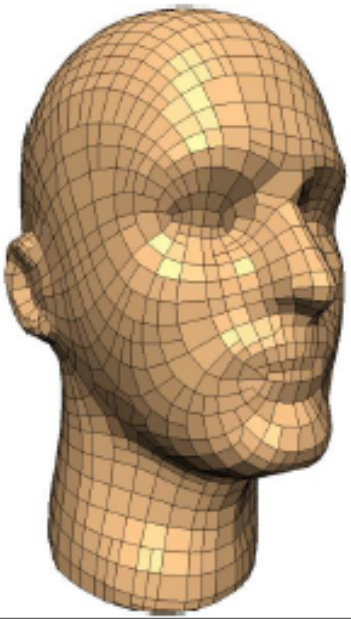
- 4 points don't always lie on a plane

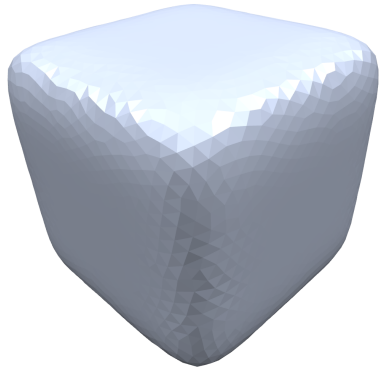


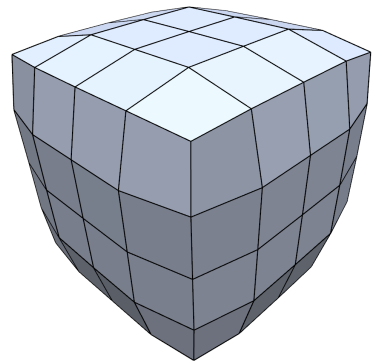
Quadrilateral meshes

Benefits of quadmeshes:

- Easier to align edges to curvature or feature lines defining an object

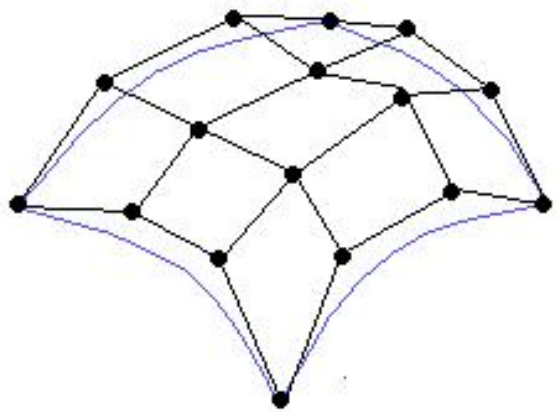






Quadrilateral meshes

- Easier to texture map
- Good for finite element simulation
- Easier to fit with parametric surfaces



Generating polygon mesh data

- Where does the mesh come from?
- Modelling using software (e.g. Blender)
- Scanning
- Procedural methods

Scanning meshes

Laser range scanning

- Measuring time for a laser beam to bounce back from surface
- Produces accurate data

Stereo vision

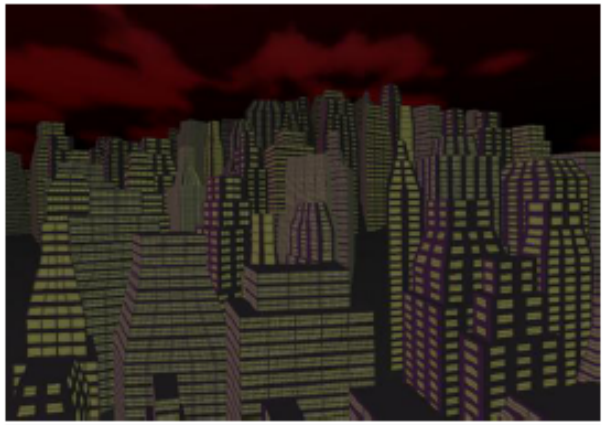
- Two cameras, estimate depth

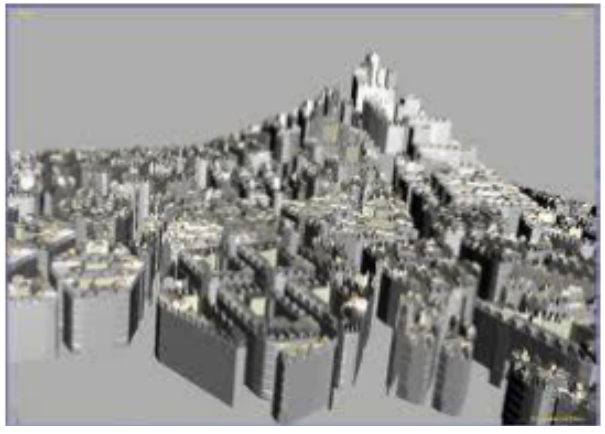
KINECT

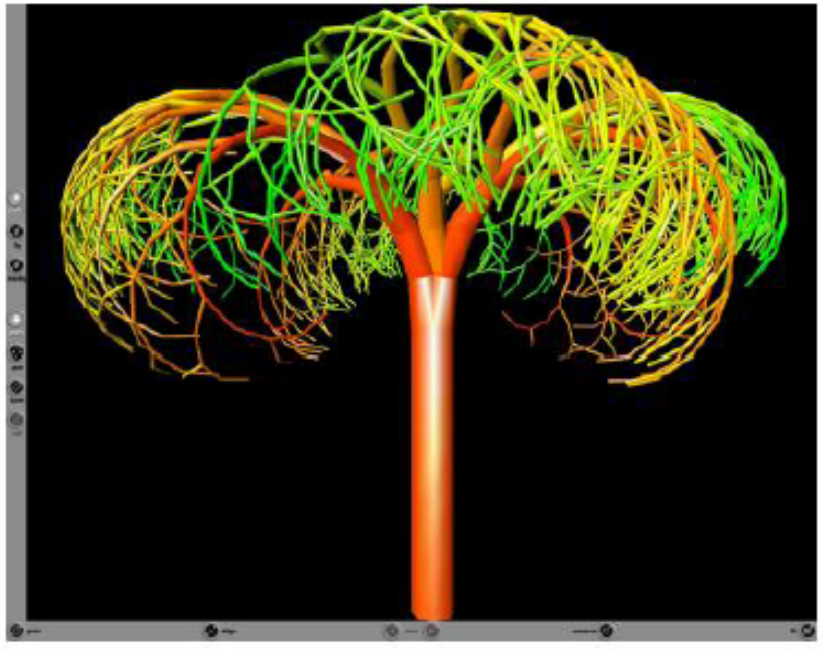
- Estimating depth from infra-red light
- Limits on range (1.2-3.5m)

Procedural models

- Rule based approaches



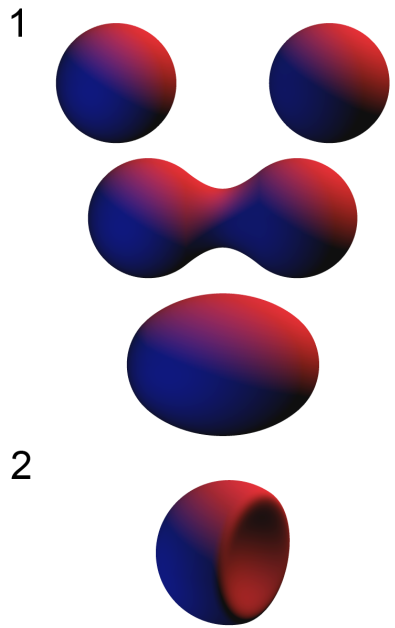




Implicit surfaces

Given some potential function $f(x)$ we can evaluate at every point in space, define an isosurface of all points where $f(x) = c$

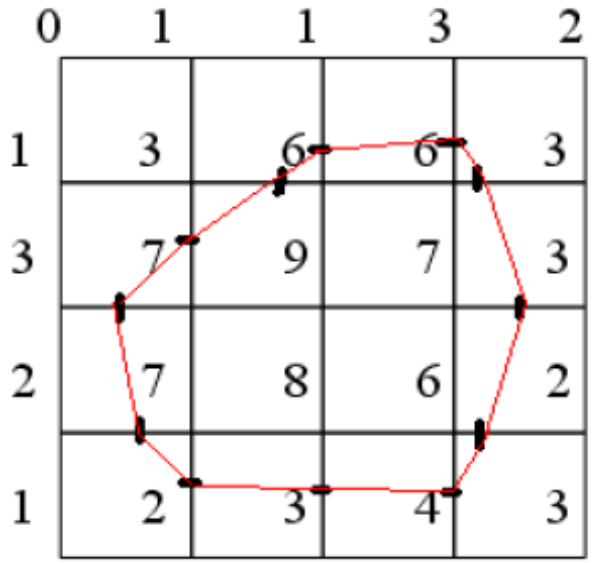
- Metaballs (Blinn, Ohmura)
- $\sum_i \frac{a_i}{r}$

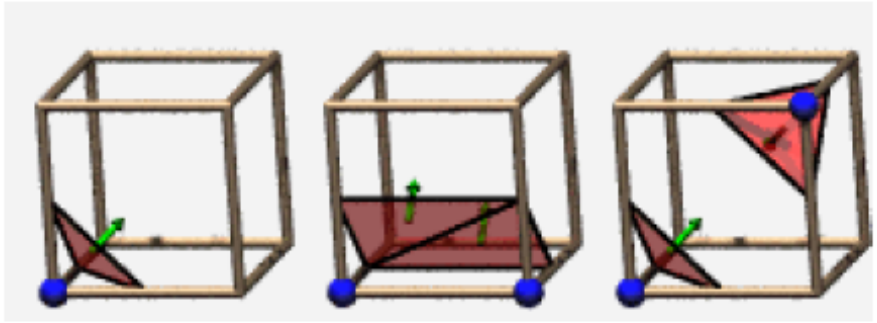


Marching cubes

Potential is computed at each point on a grid

- where edges of the grid cross the threshold, the surface is produced
- set of rules for producing the surface based on which edges cross the threshold





Overview

Polygon meshes

- Storage
- Representation
- Decomposition
- Quad meshes
- Generation
- Implicit surfaces

Level of detail

- Scaling methods

Image representations

- Pixel Buffers
- Alpha channels
- Z buffers

Level of detail scaling

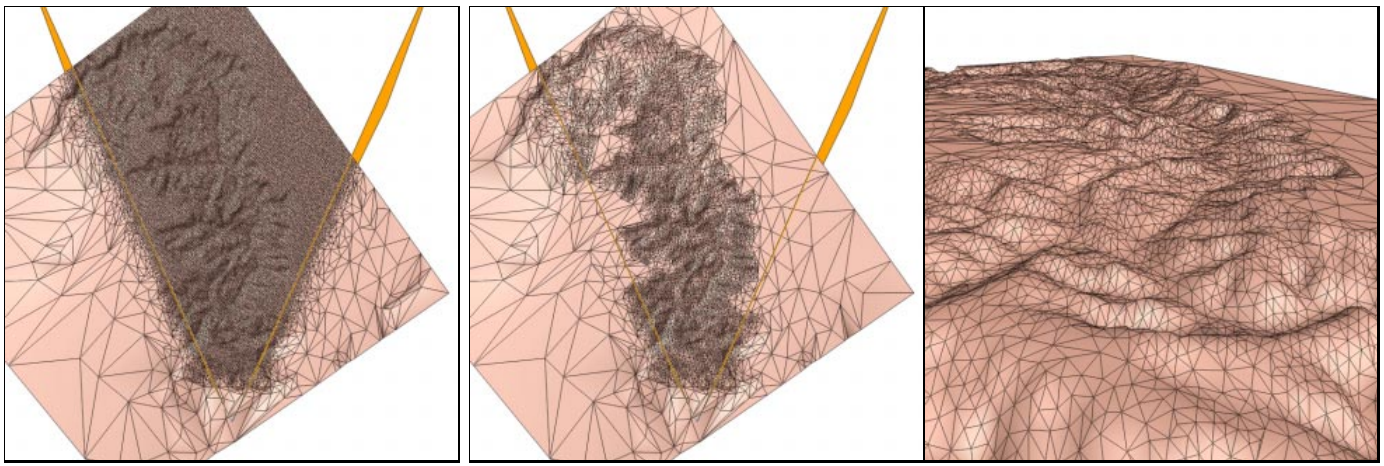
When an object is close to the camera we usually want a very detailed model. But not when:

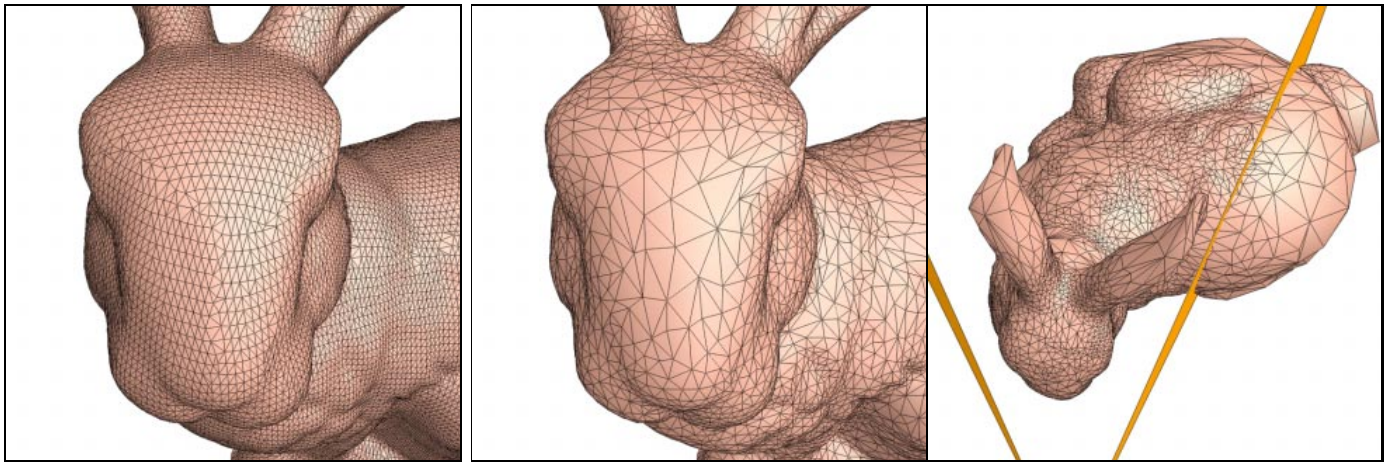
- further away (taking up a small amount of space on screen)
- facing away from the camera
- outside the view volume

Level of detail scaling

Progressive meshes

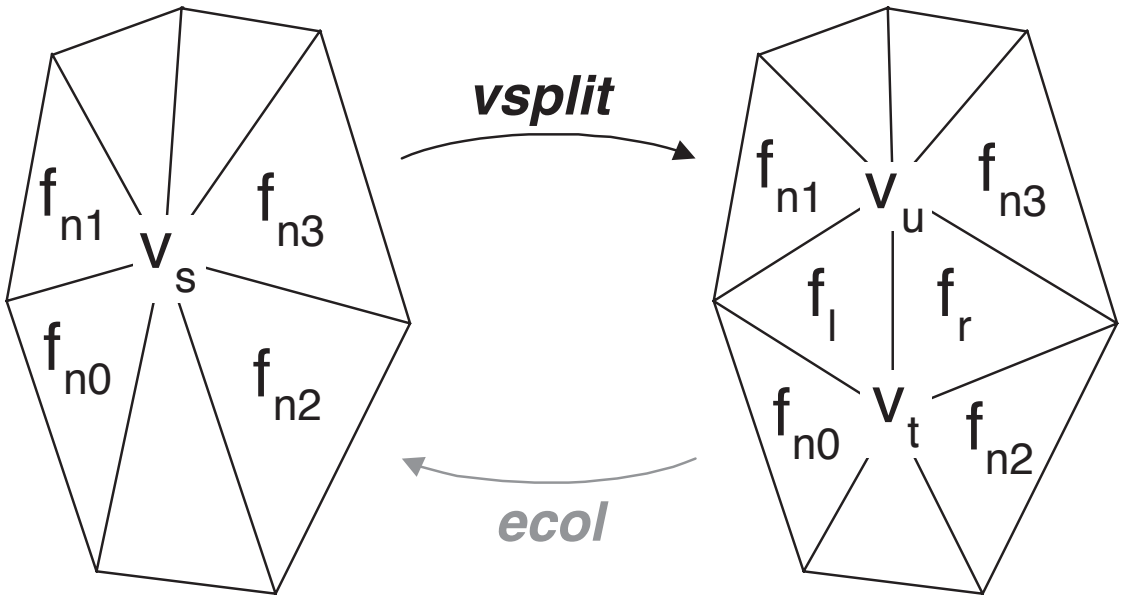
- some polygons may be much closer to camera than others, within a single mesh





Level of detail scaling

Progressive meshes



Overview

Polygon meshes

- Storage
- Representation
- Decomposition
- Quad meshes
- Generation
- Implicit surfaces

Level of detail

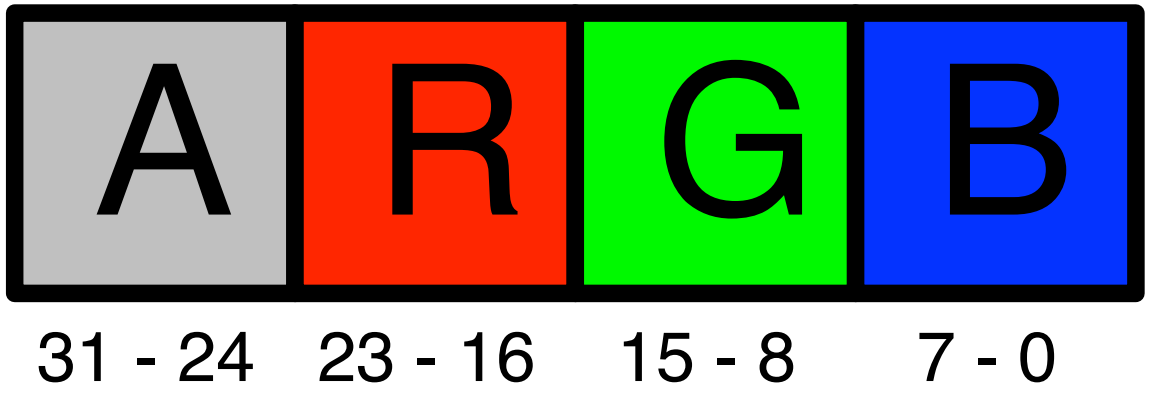
- Scaling methods

Image representations

- Pixel Buffers
- Alpha channels
- Z buffers

Image representations

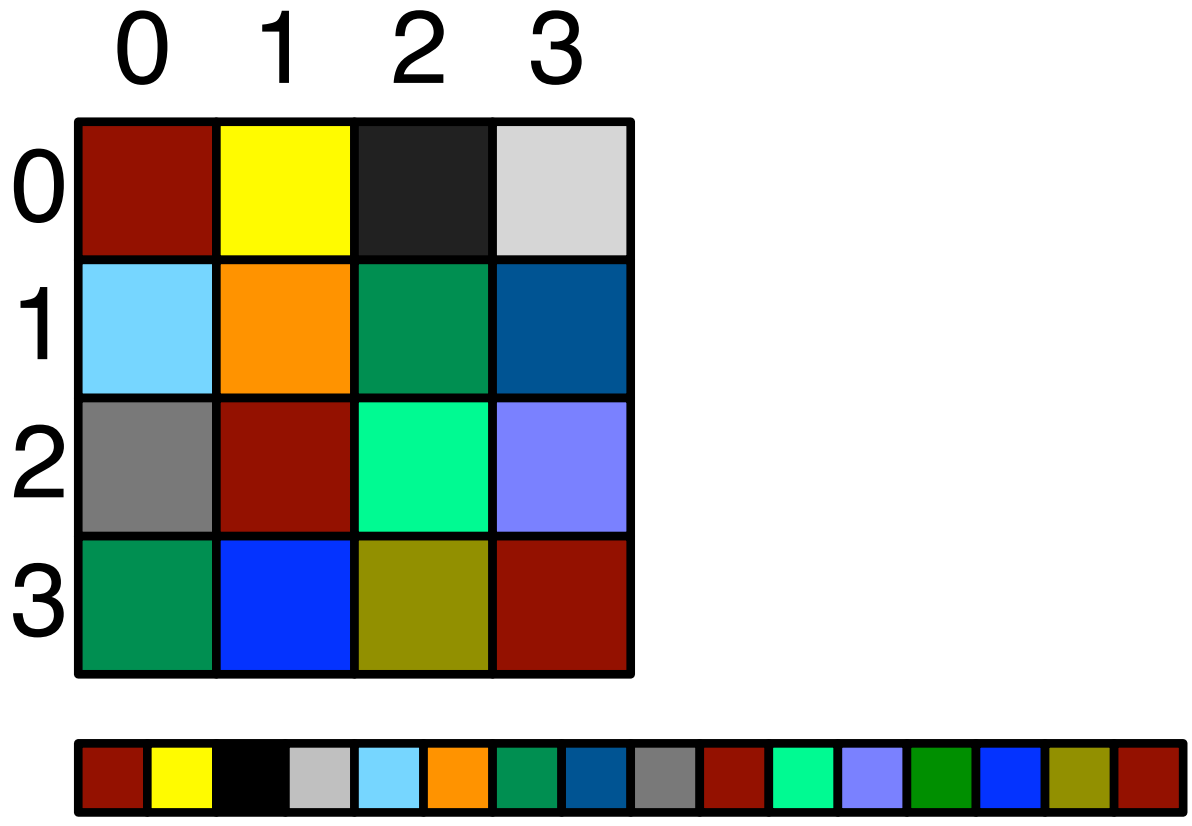
Standard 32 bit format for colours:



Other formats:

- 32bit Floats per channel (high dynamic range)
- 24bit RGB (usually padded)

Pixel buffers



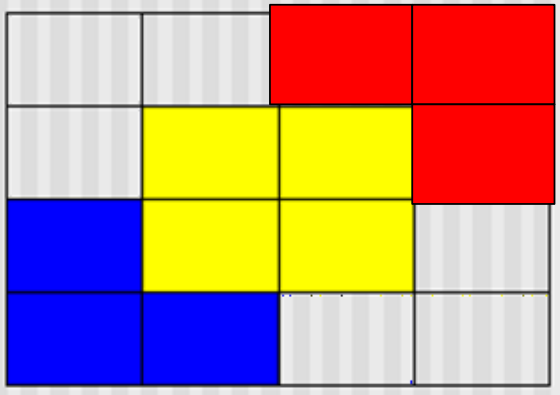

```
int array[4][4];
int arrayFlat[16];
array[y][x]=arrayFlat[y*width+x];
```

Z buffers

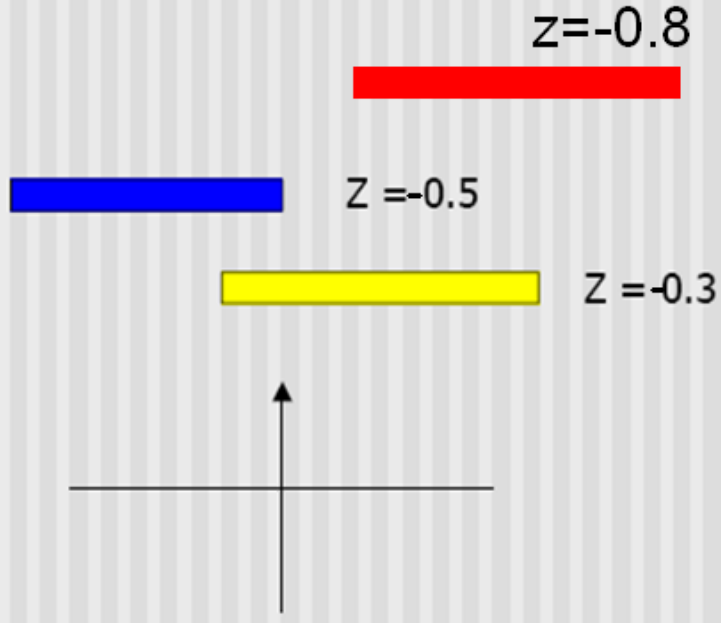
How do we make sure things closer to the camera are drawn on top of things behind?

- various methods covered later in the course
- current most popular method is the Z buffer
- implemented by GPU, very little work to use with OpenGL

Z buffer example



Correct Final image



Top View

Z buffer example

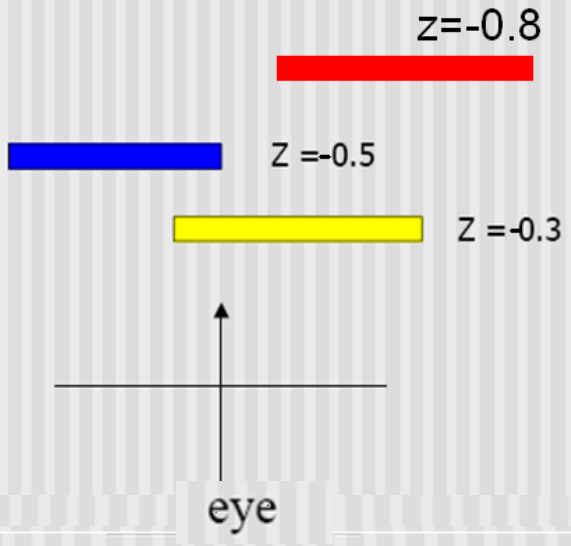
Step 1: Initialize the depth buffer

-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0

Z buffer example

Step 2: Draw the blue polygon (assuming the program draws blue polyon first – the order does not affect the final result any way).

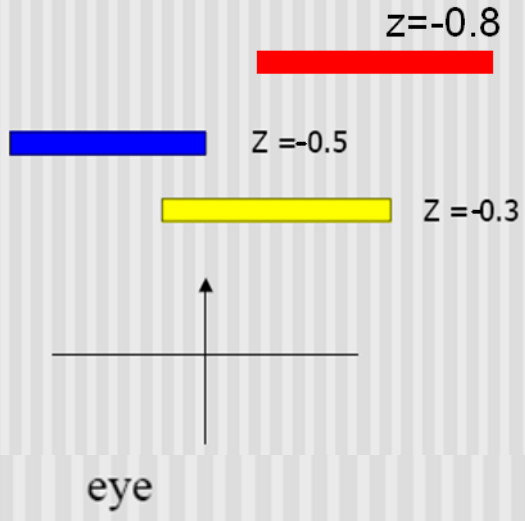
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-0.5	-0.5	-1.0	-1.0
-0.5	-0.5	-1.0	-1.0



Z buffer example

Step 3: Draw the yellow polygon

-1.0	-1.0	-1.0	-1.0
-1.0	-0.3	-0.3	-1.0
-0.5	-0.3	-0.3	-1.0
-0.5	-0.5	-1.0	-1.0

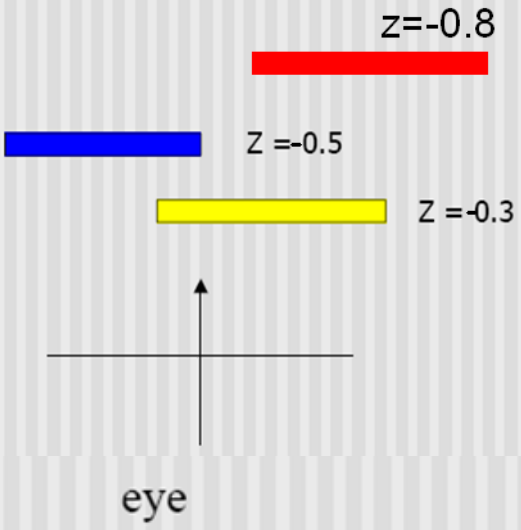


If the depth value is larger than that in the z-buffer, the pixel is coloured and value in the z-buffer is updated

Z buffer example

Step 4: Draw the red polygon

-1.0	-1.0	-0.8	-0.8
-1.0	-0.3	-0.3	-0.8
-0.5	-0.3	-0.3	-1.0
-0.5	-0.5	-1.0	-1.0



If the depth value is larger than that in the z-buffer, the pixel is coloured and value in the z-buffer is updated

Summary

Polygon meshes

- How to store them
- Data structures for operations on meshes
- Creating mesh data

Level of detail

- Why?
- Scaling

Image data

- Standard storage format
- Alpha channels
- Z buffers

References

Shirley:

- Chapter 12.1 (Data Structures for Graphics – Triangle Meshes)
- Chapter 3 (Raster Images)

Papers:

- Taubin, G., & Rossignac, J. (1998). Geometric compression through topological surgery. ACM Transactions on Graphics, 17(2), 84–115.
- D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini and D. Zorin, Eurographics STARS, 2012
- Hoppe, H. (1997). View-dependent refinement of progressive meshes (pp. 189–198). SIGGRAPH '97