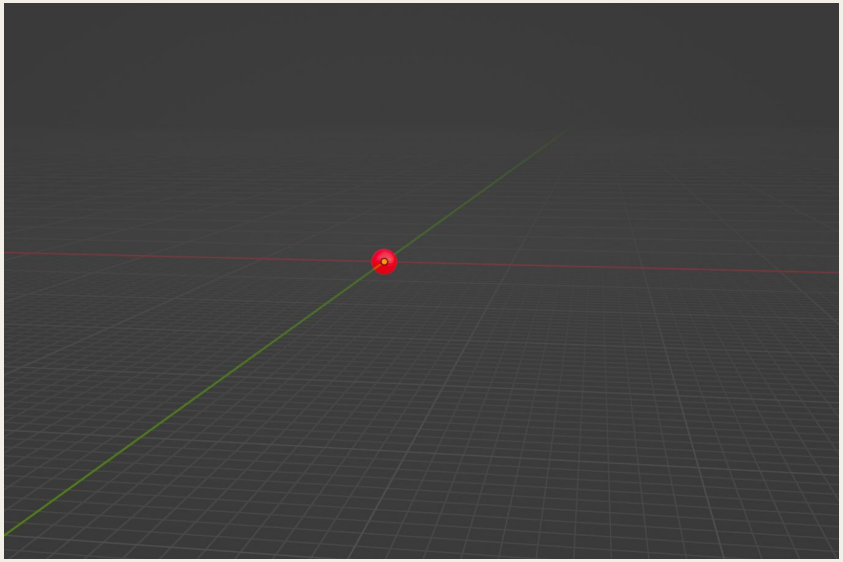
CS148 Summer 2025 - Lecture 2

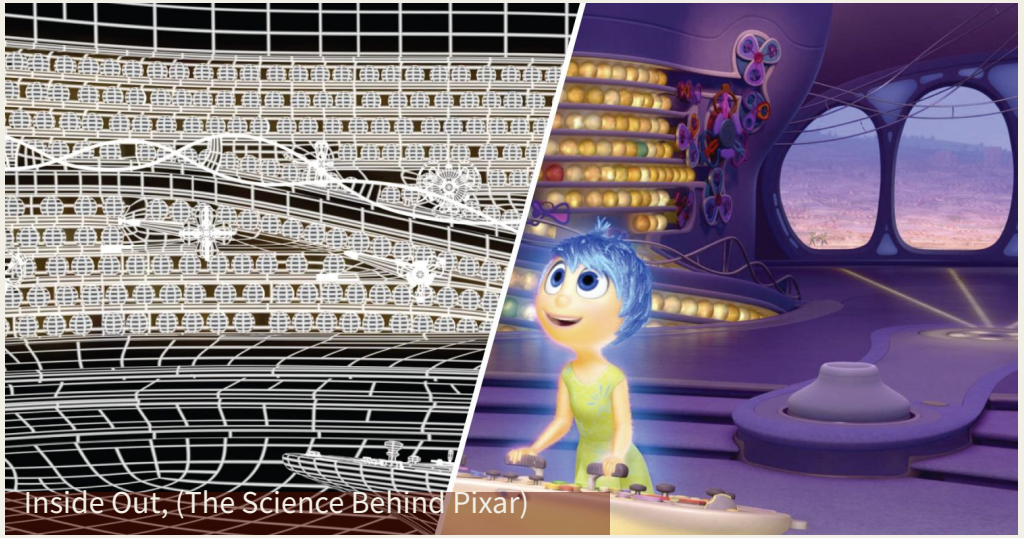
06/26/2025

Geometry and Transformations

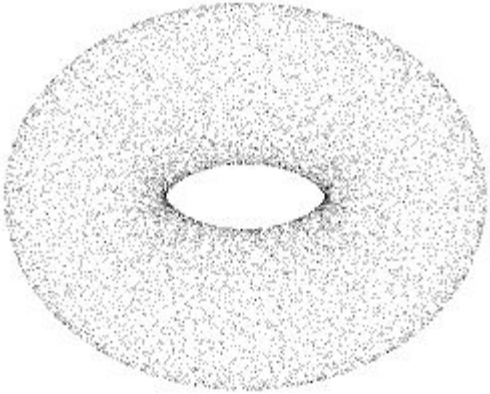
cs148.stanford.edu

Points in Virtual Space

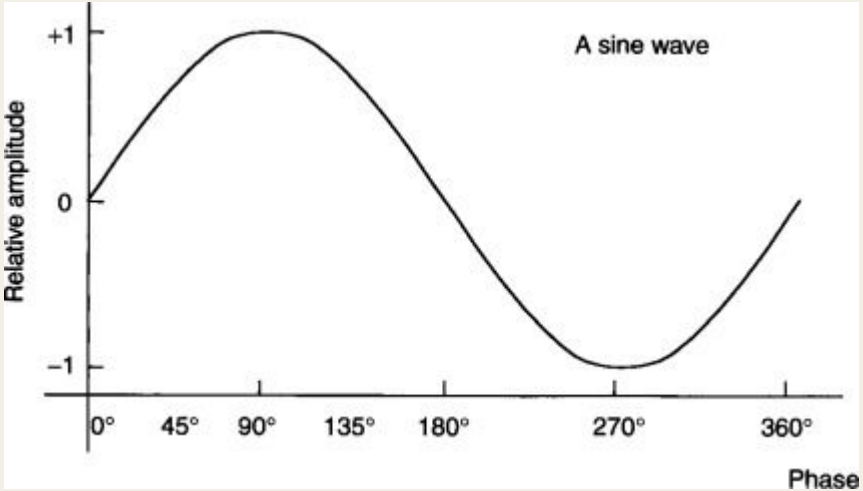


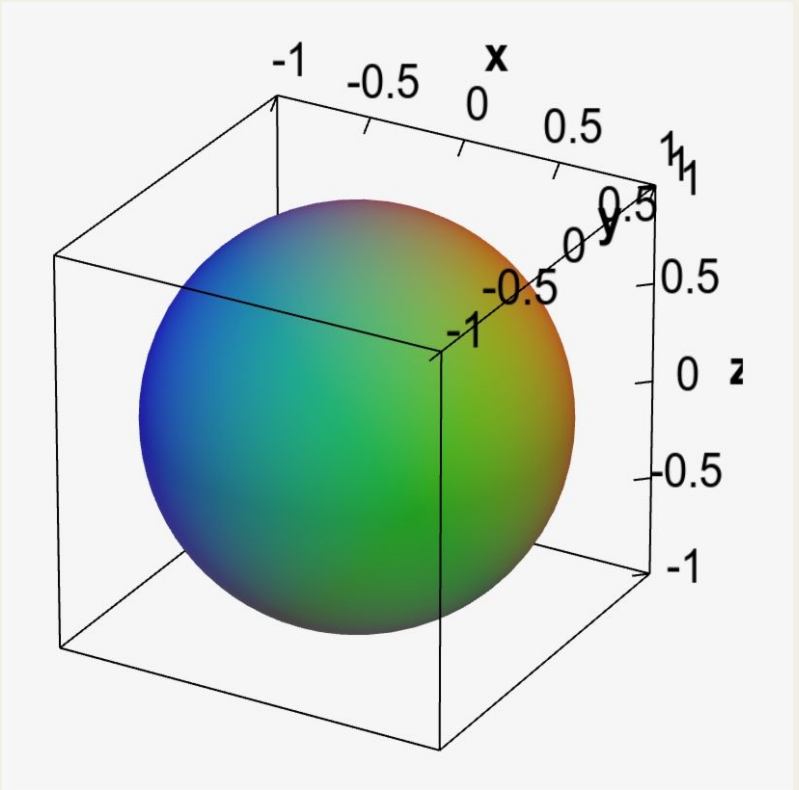


Point Clouds

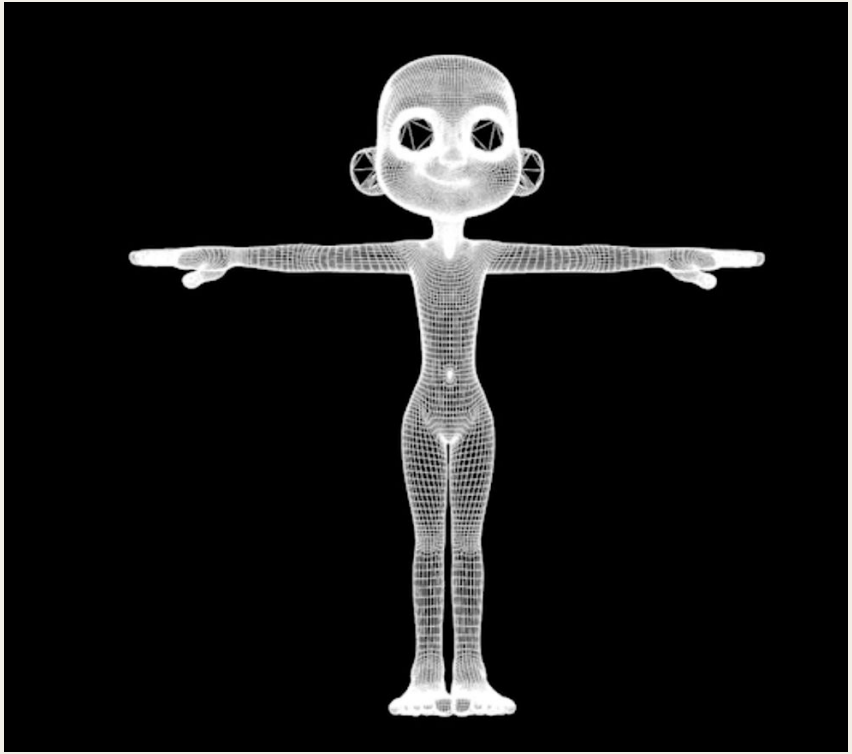


Implicit Surfaces

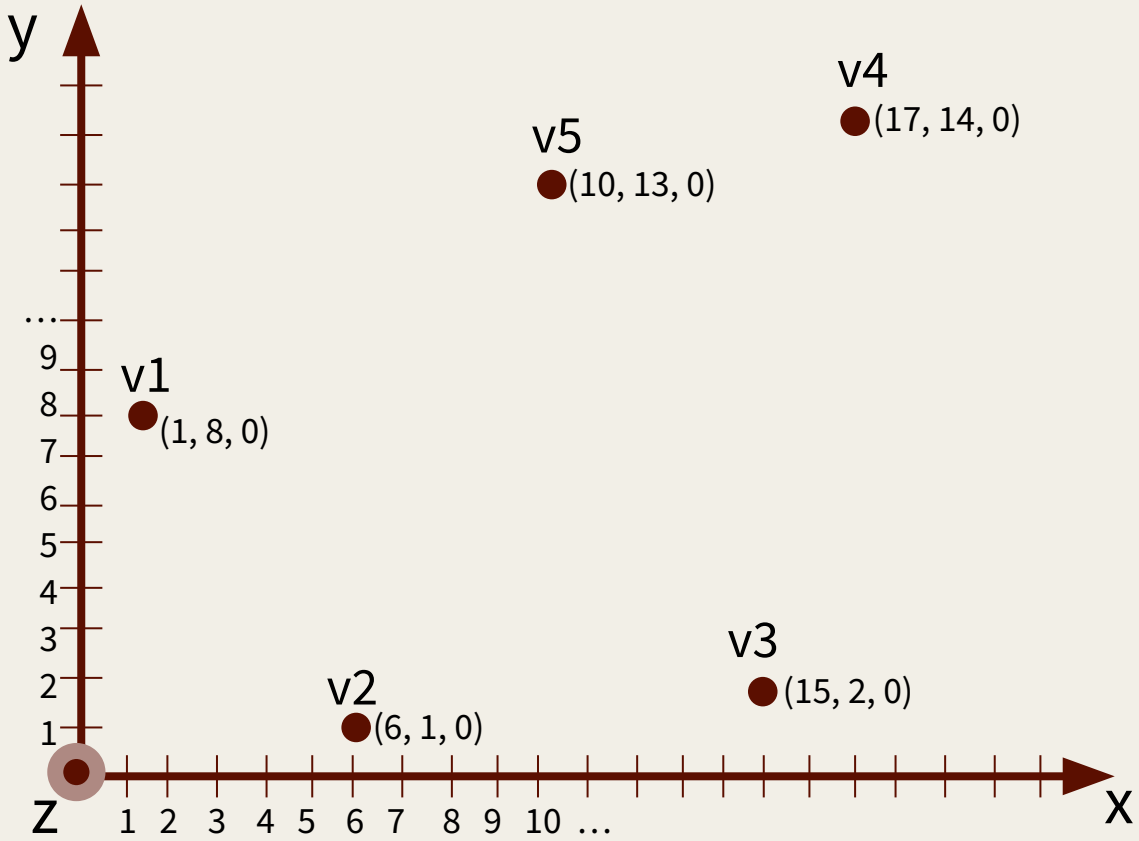




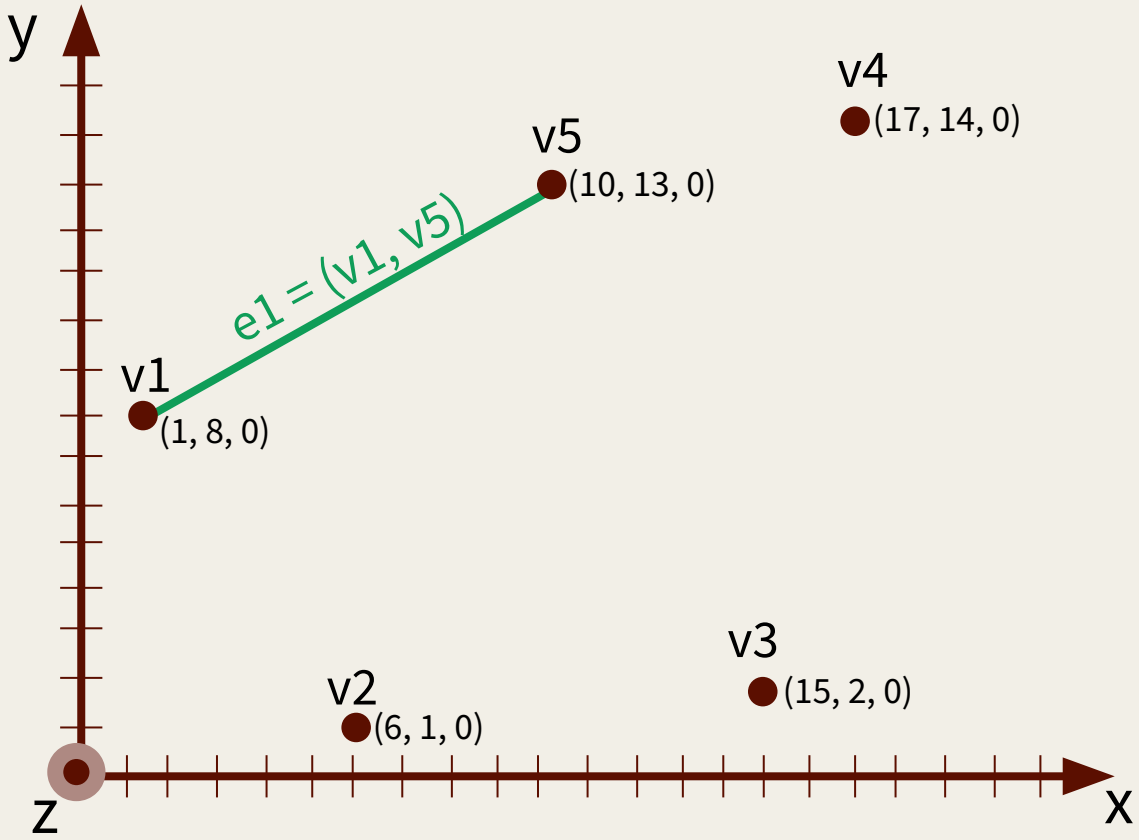
Mesh Surfaces



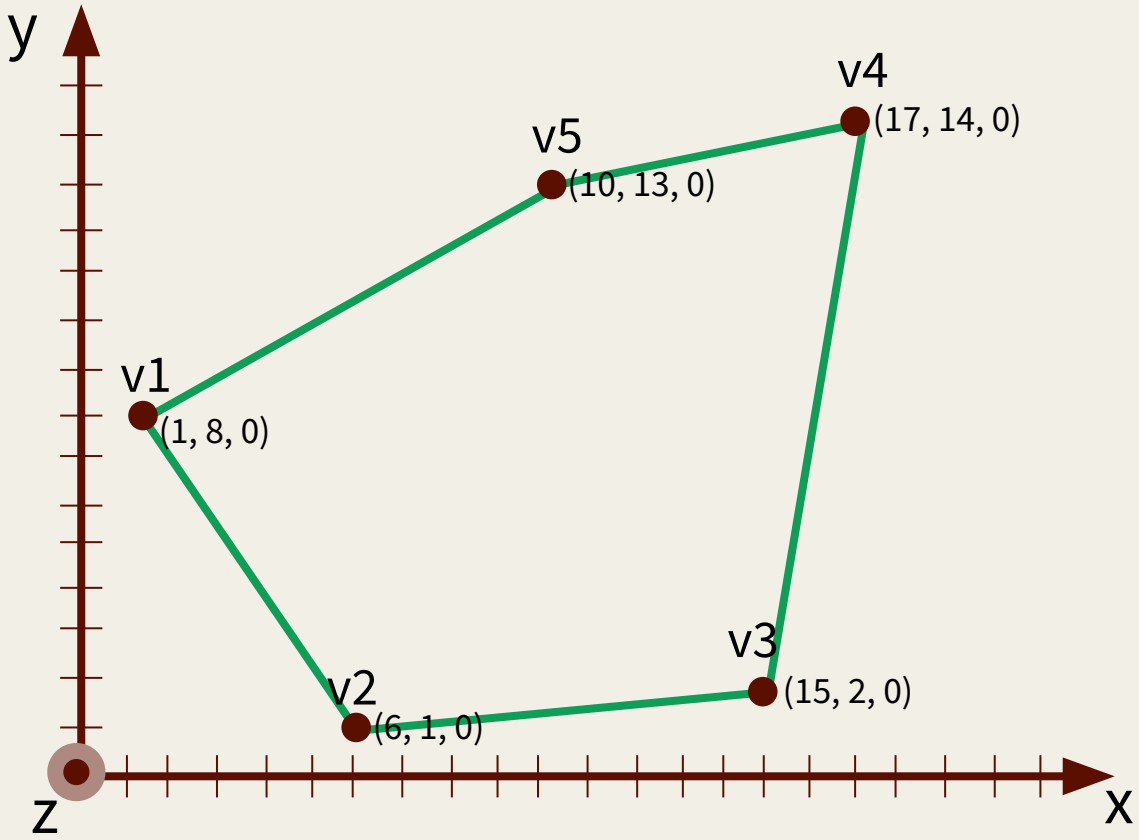
Points, Edges, and Faces



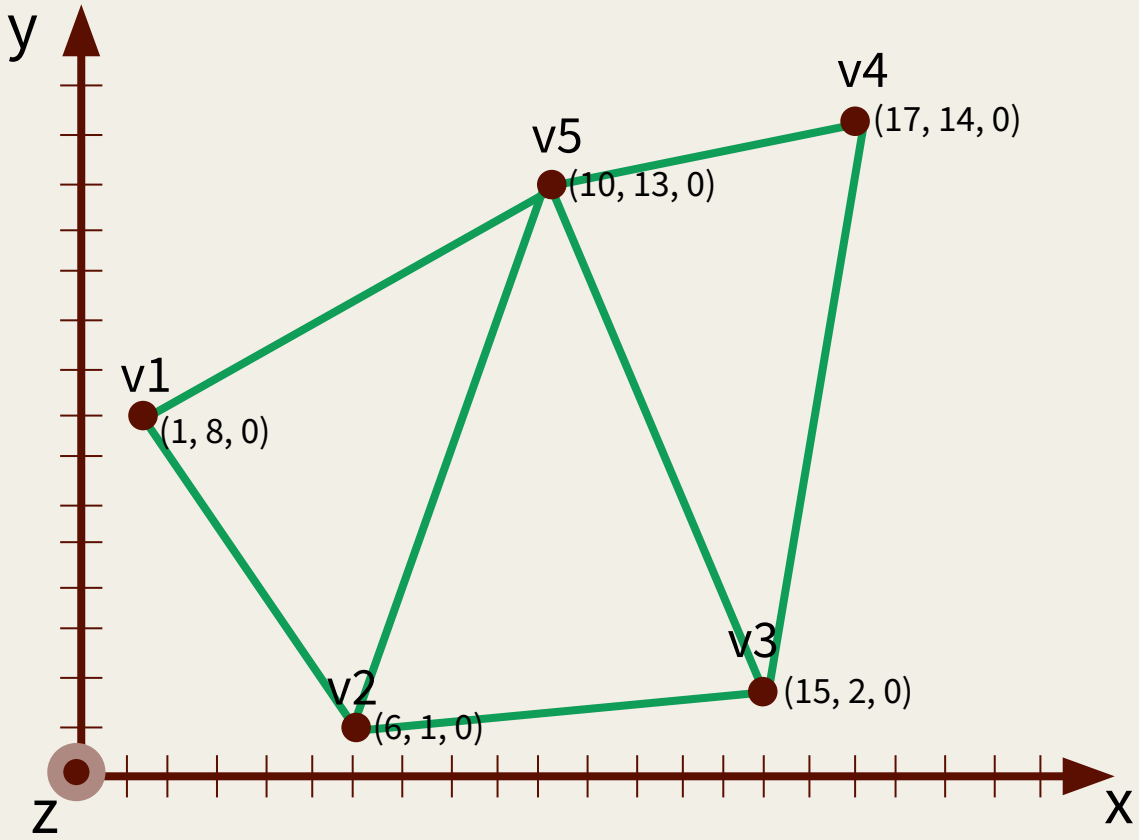
Points, Edges, and Faces

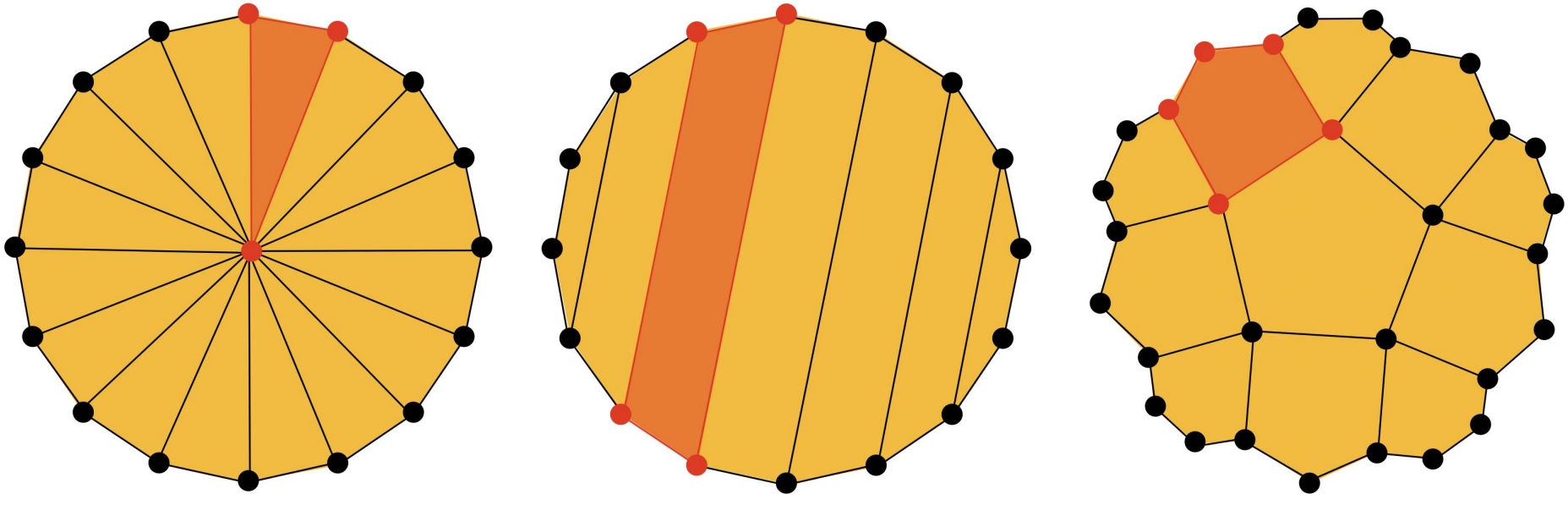


Points, Edges, and Faces



Points, Edges, and Faces

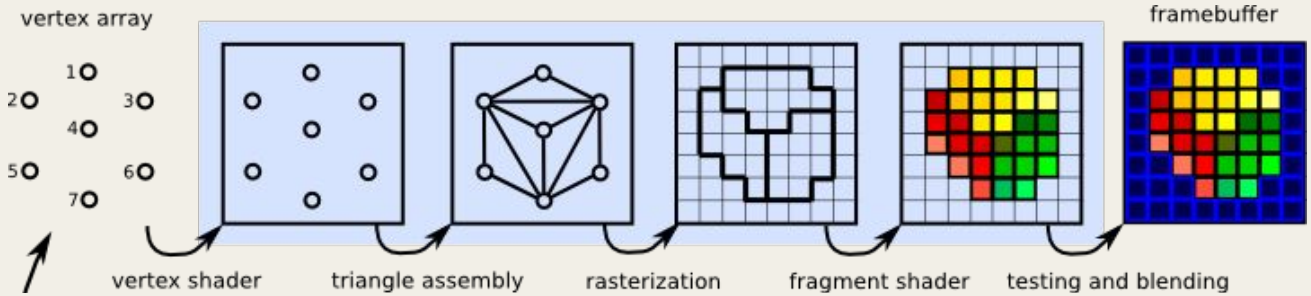


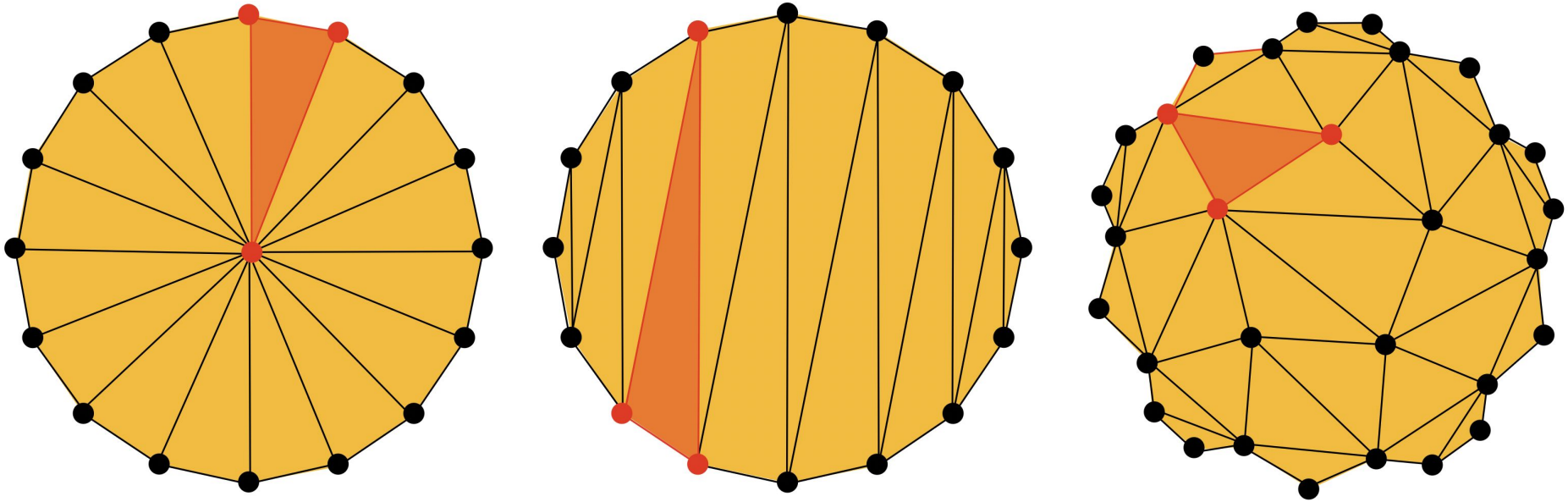


OpenGL and the Graphics Pipeline

- Blender uses OpenGL for it's real-time scanline renderer

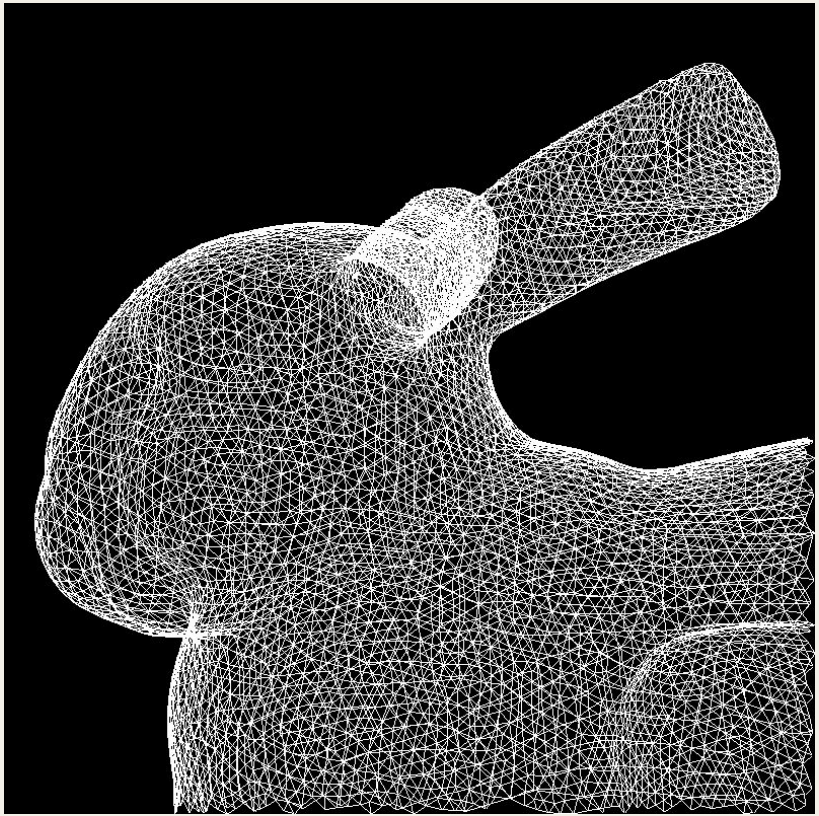
- OpenGL started by Silicon Graphics Inc. (SGI) 1991 (public 2006)
- Drawing API for 2D/3D graphics
- Designed to be implemented mostly on hardware
- Main competitor is DirectX
- Highly optimized for triangles





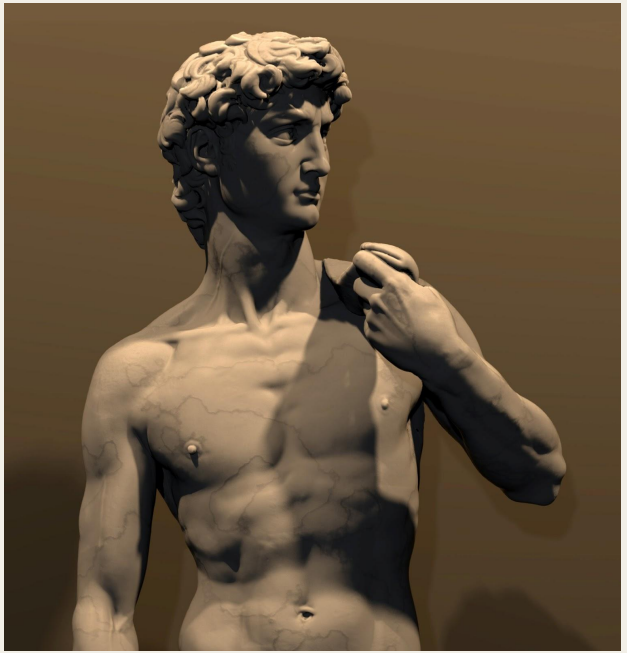
Why Triangles?

- Can optimize and specialize the geometry pipeline for 1 shape
- Software and algorithms can be optimized
- Hardware (e.g. GPUs) can be specialized



Lots of Triangles

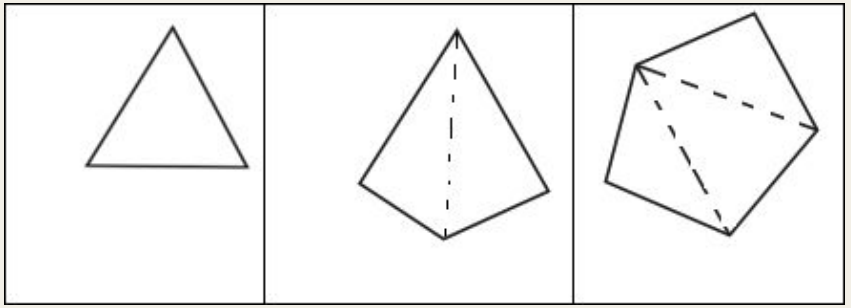
Stanford Bunny
69,451 triangles



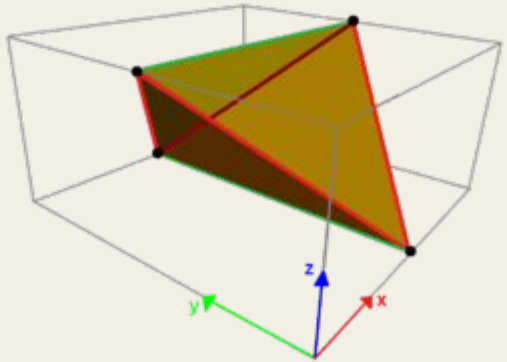
David (Digital Michelangelo Project)
56,230,343 triangles

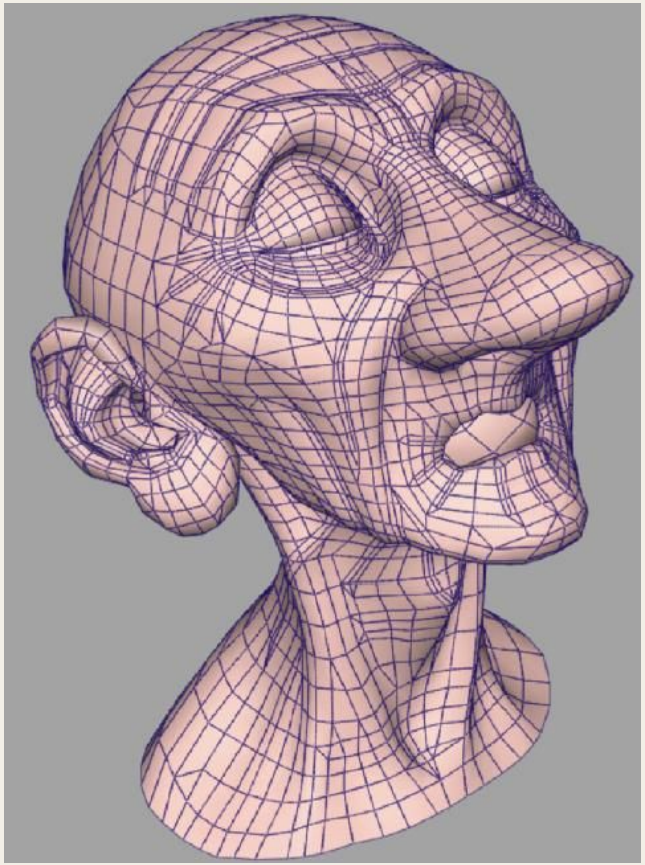
Mathematical Advantages of Triangles

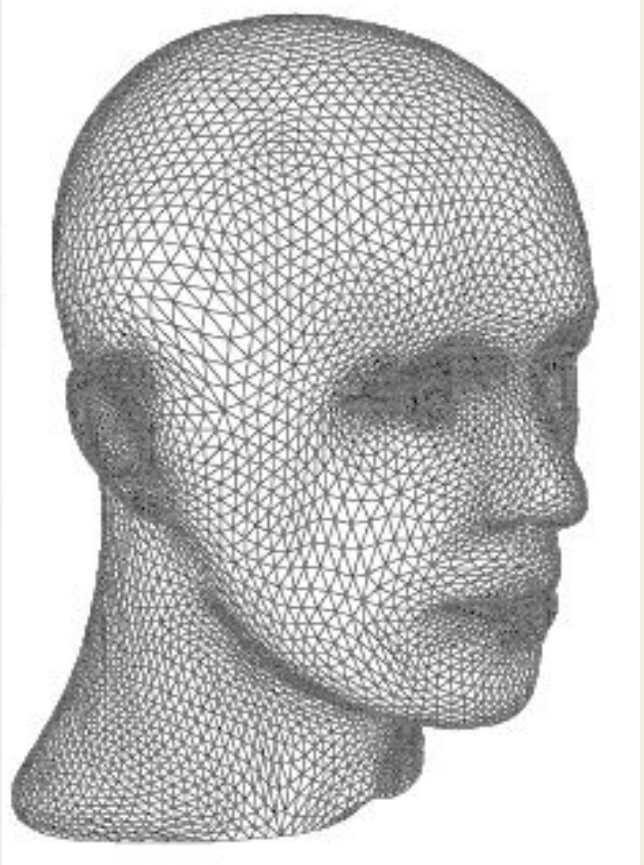
- Easy to break other polygons into triangles
- Triangles guaranteed planar
- Complex objects are well approximated with triangles
- Geometric transformations only need to be applied to vertices

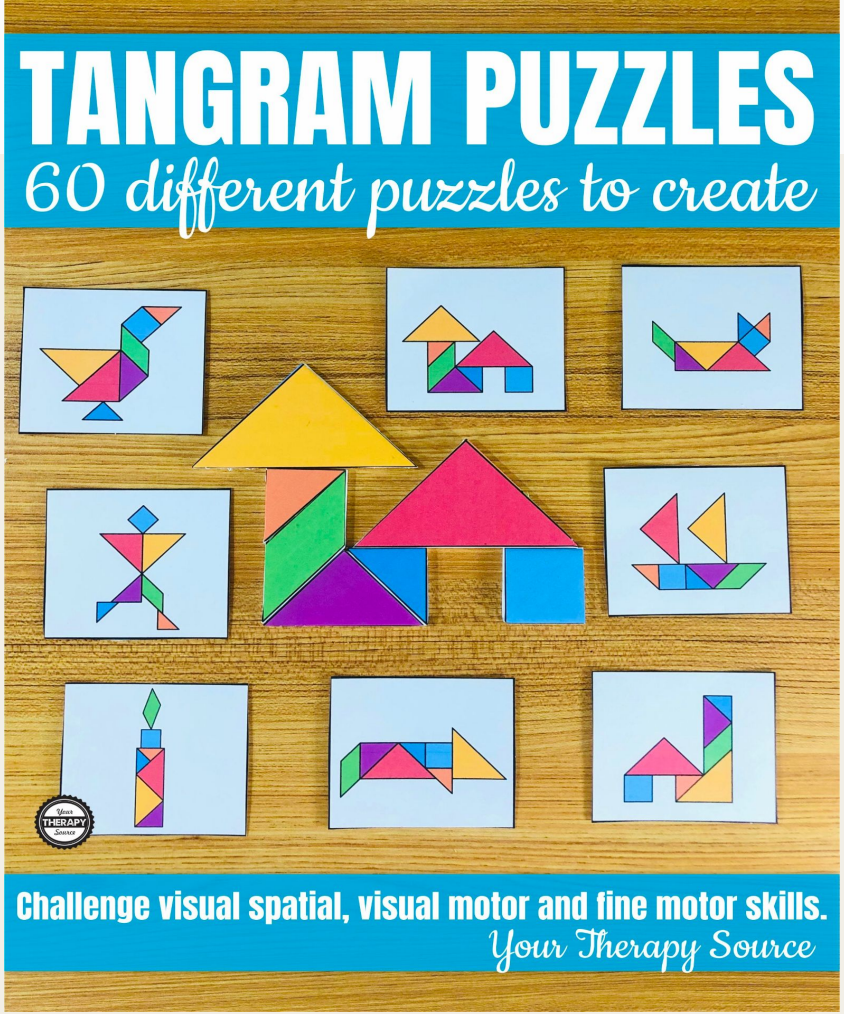


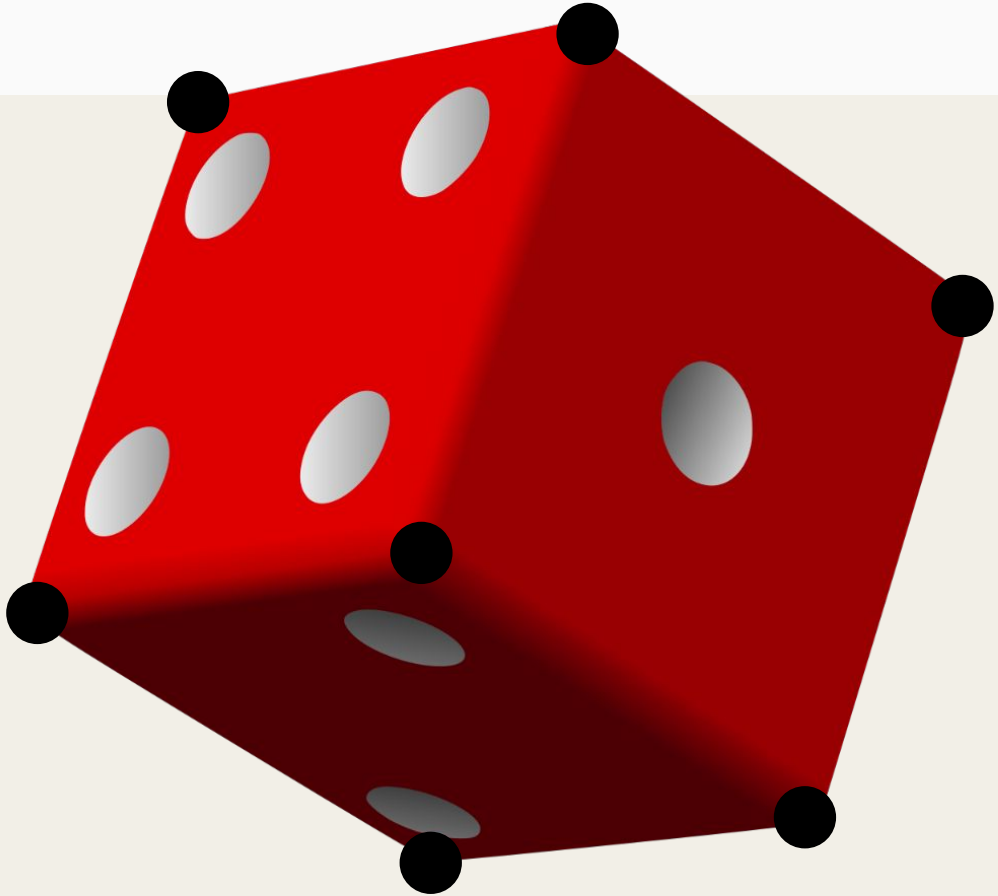
- More, e.g. barycentric interpolation to interpolate vertices to interior
- Not everything though, e.g. fluid sim

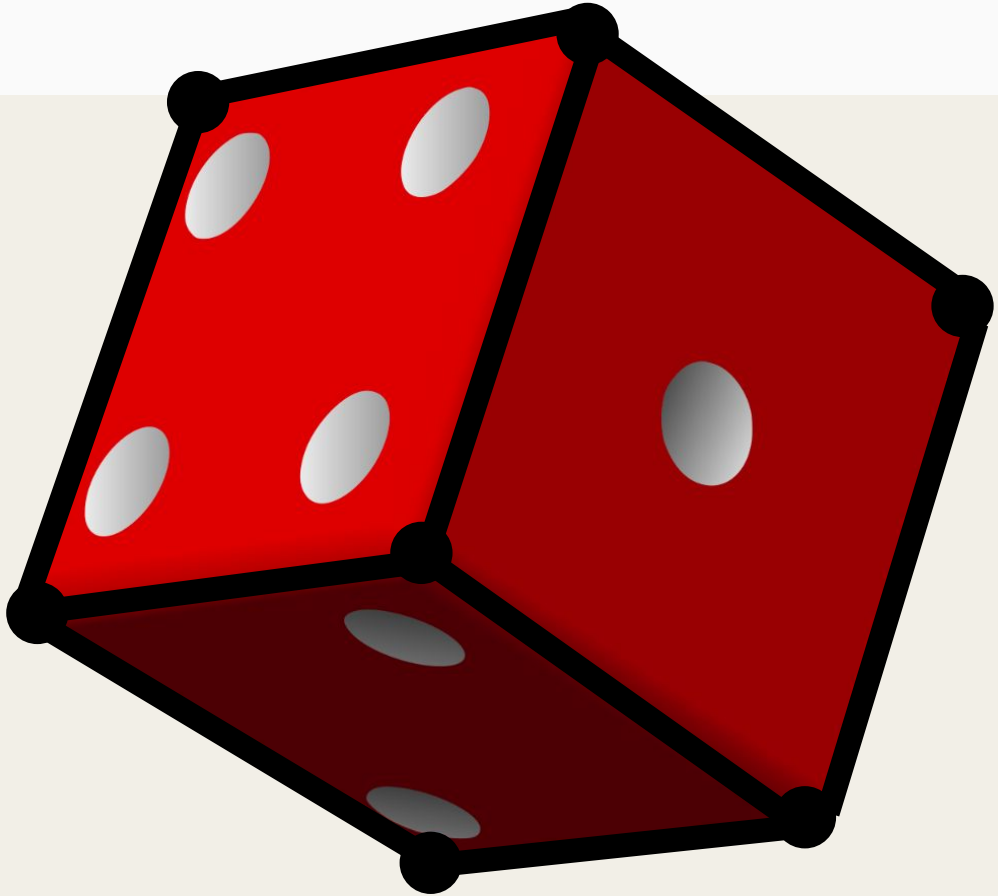


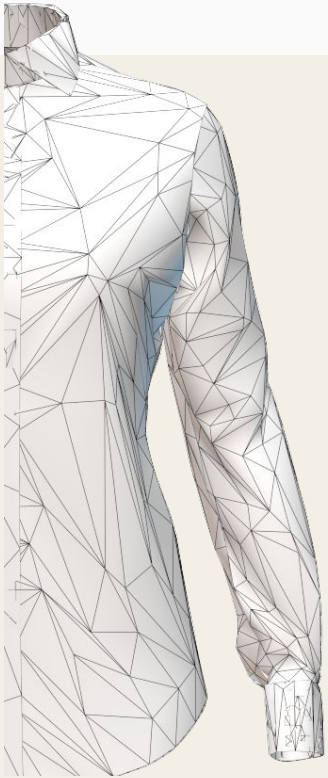


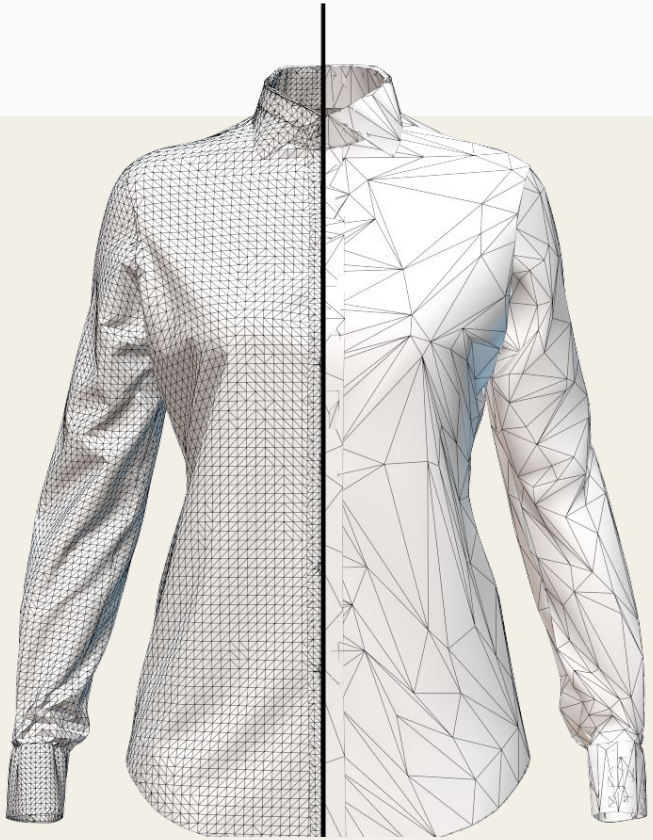



TANGRAM PUZZLES
60 different puzzles to create
Challenge visual spatial, visual motor and fine motor skills.
Your Therapy Source









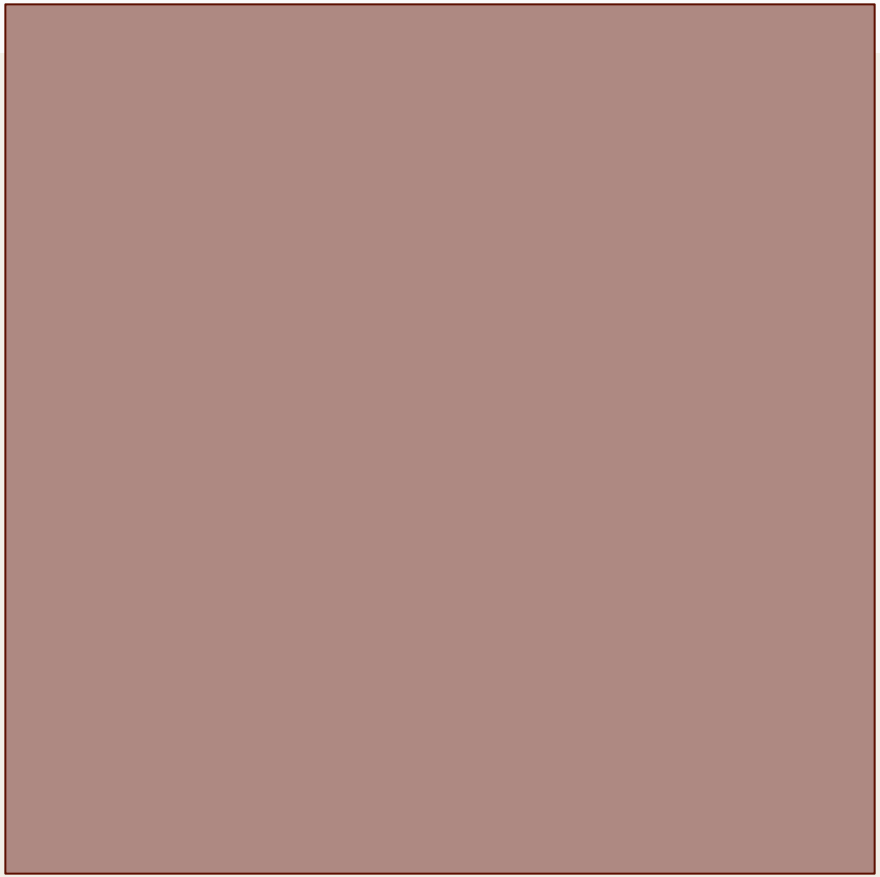
OBJ Files

- Developed by Wavefront Technologies ~1990
- Openly documented ~1995
- Now one of the universal geometry definition file formats used by graphics applications

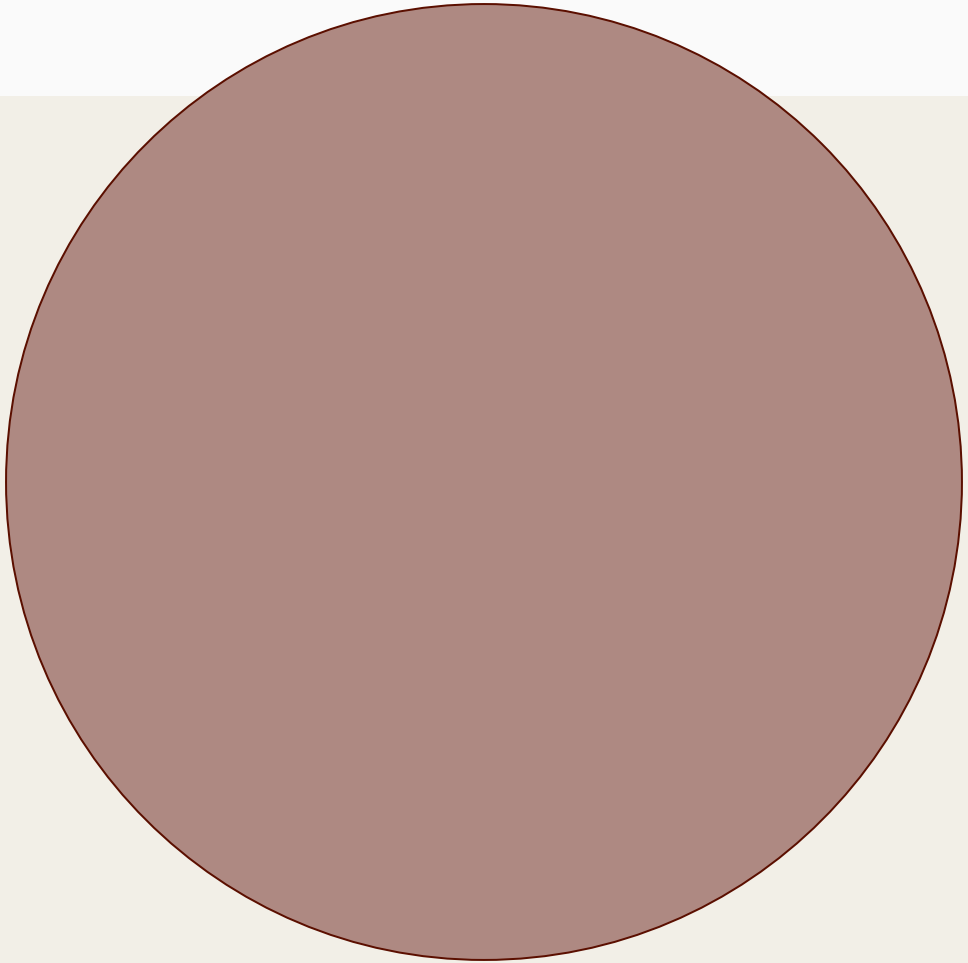
- Most basic form: list of vertices and faces
- “v” lines denote x, y, z coordinates of vertices
- “f” lines denote indices of vertices for the face
- Indices 1 indexed

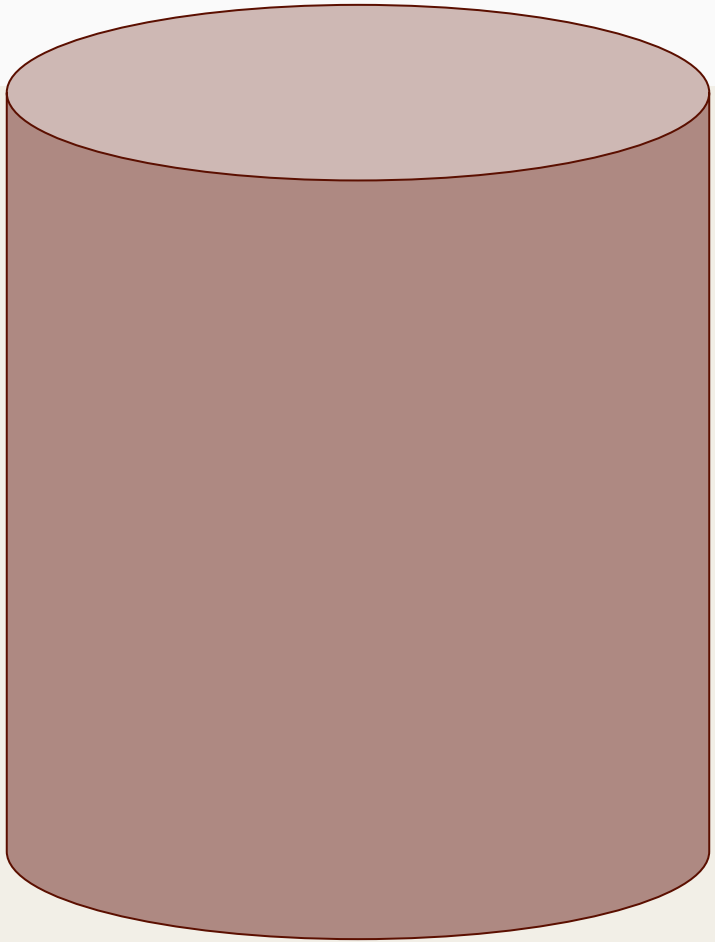
- Also supports other polygon meshes

```
v x1 y1 z1
v x2 y2 z2
v x3 y3 z3
...
v xm ym zm
f face0v1 face0v2 face0v3
f face1v1 face1v2 face1v3
f face2v1 face2v2 face2v3
...
f facenv1 facenv2 facenv3
```



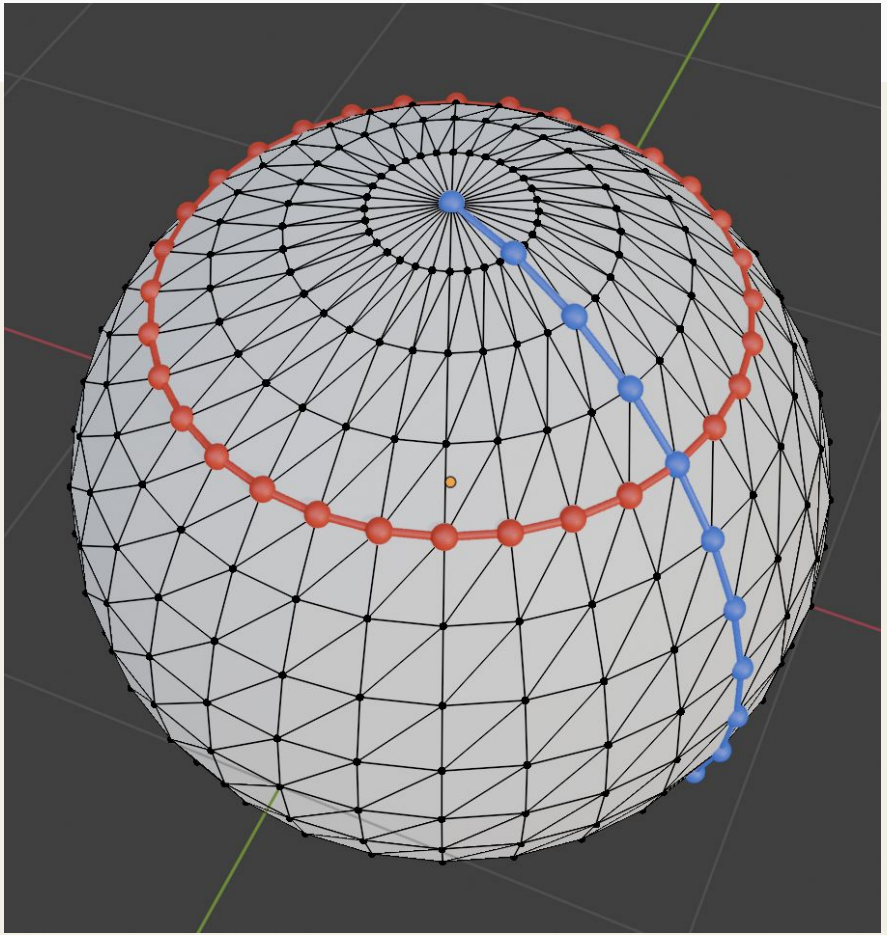
```
v x1 y1 z1
v x2 y2 z2
v x3 y3 z3
...
v xm ym zm
f face0v1 face0v2 face0v3
f face1v1 face1v2 face1v3
f face2v1 face2v2 face2v3
...
f facenv1 facenv2 facenv3
```





Tessellating a Sphere

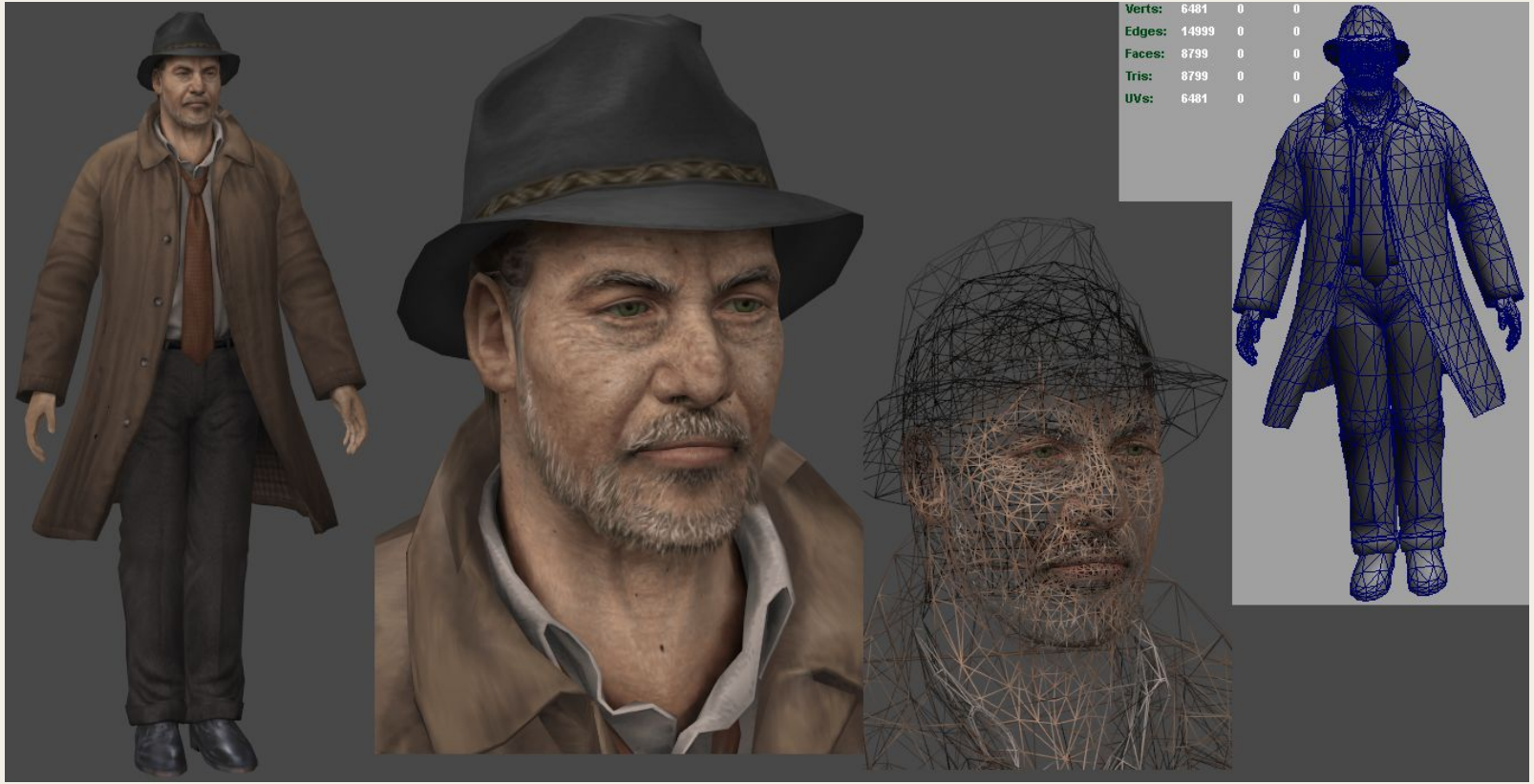
- Exercise on Homework

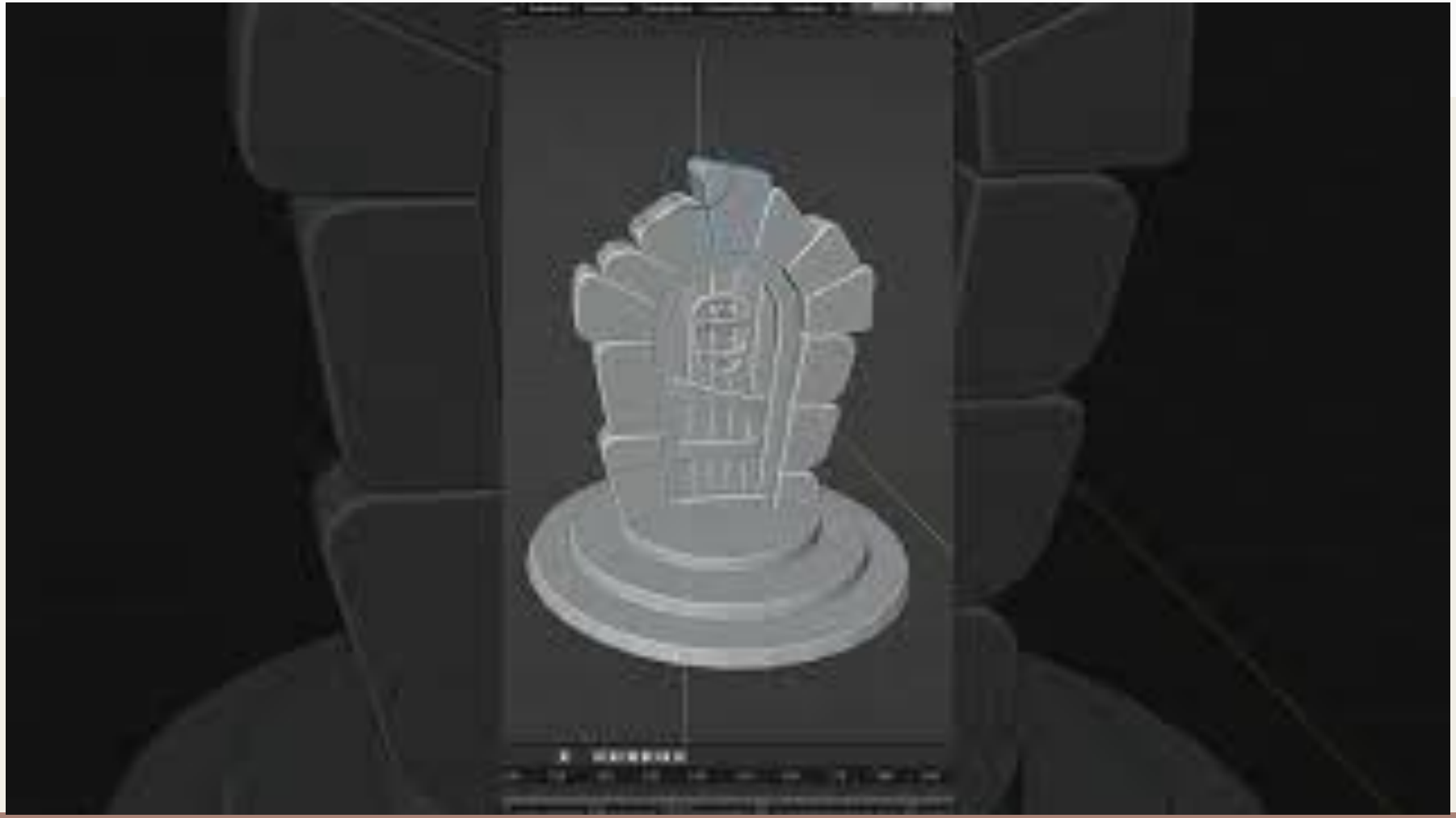


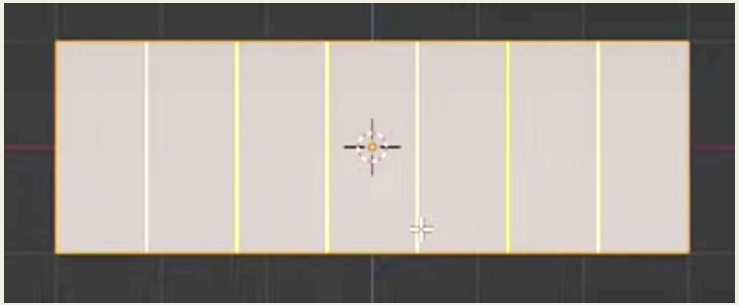
Example Objects

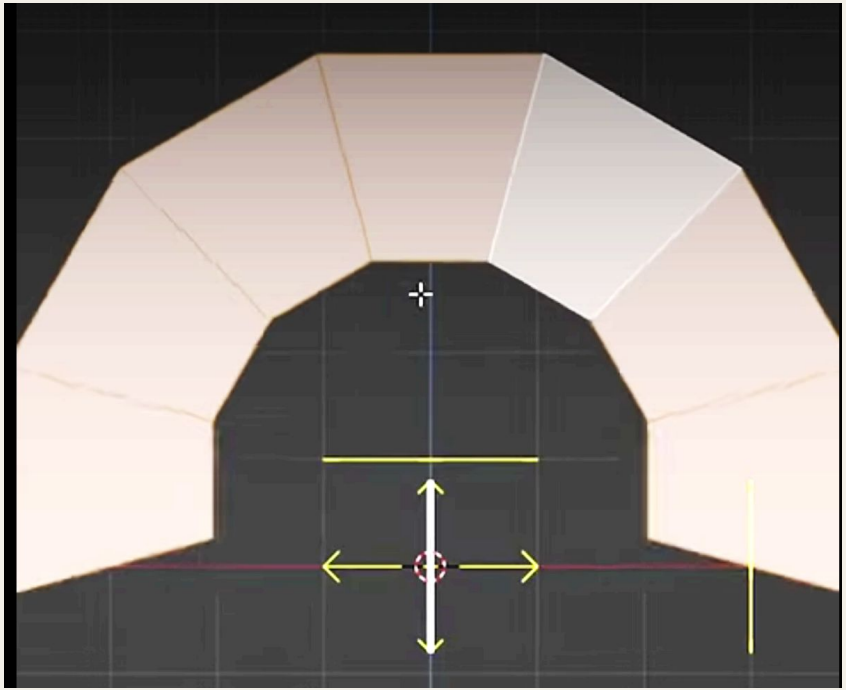
Example: Movies

Example: Games









Activity: Make your own object

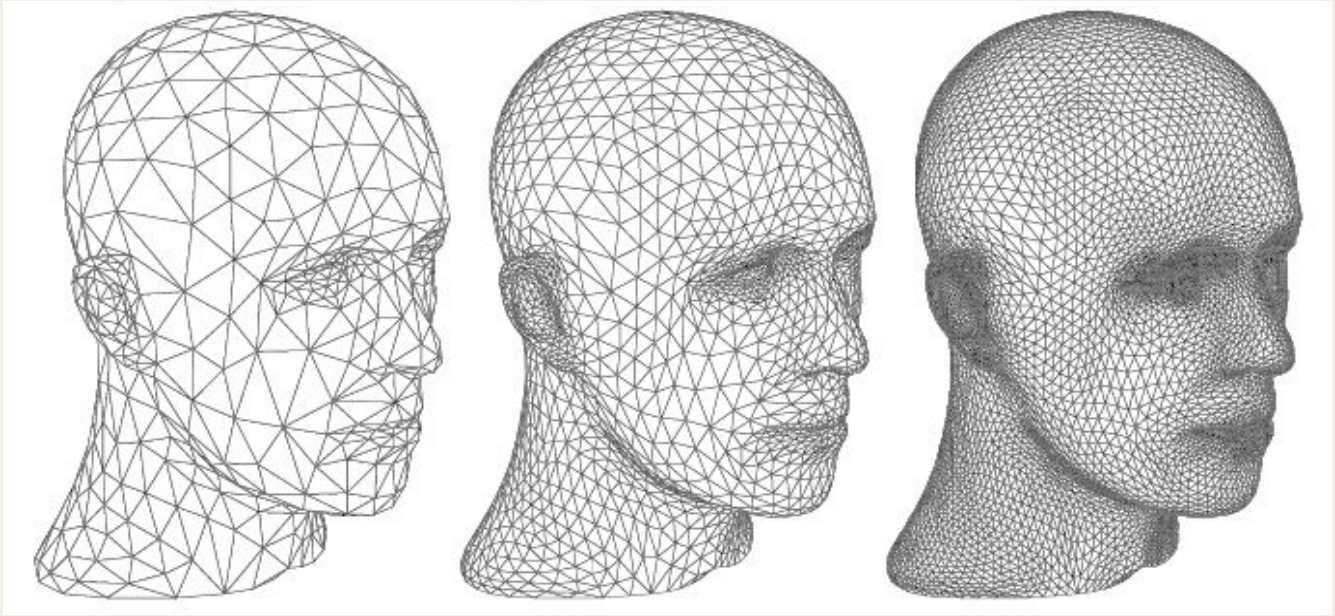
Activity: Make your own object

- Get into groups
- Find an object you might reproduce virtually
- Discuss where vertices and edges might go - what shapes is this object made of?

Questions?

Adding Geometry: Subdivision

- Procedural algorithm: automatically generate a finer/smoother mesh from a coarser mesh



General Idea with Curves

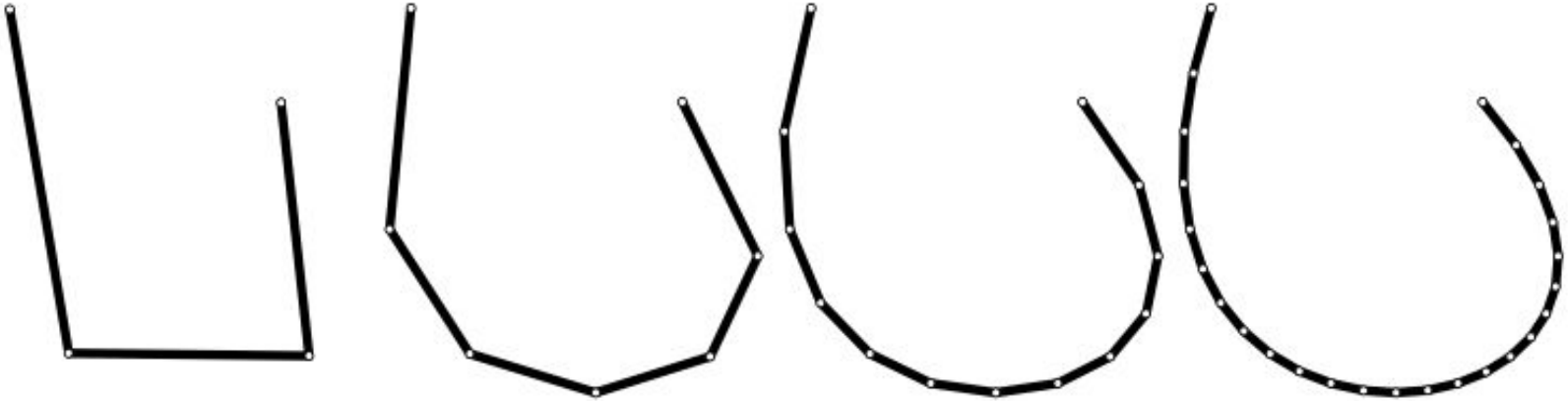
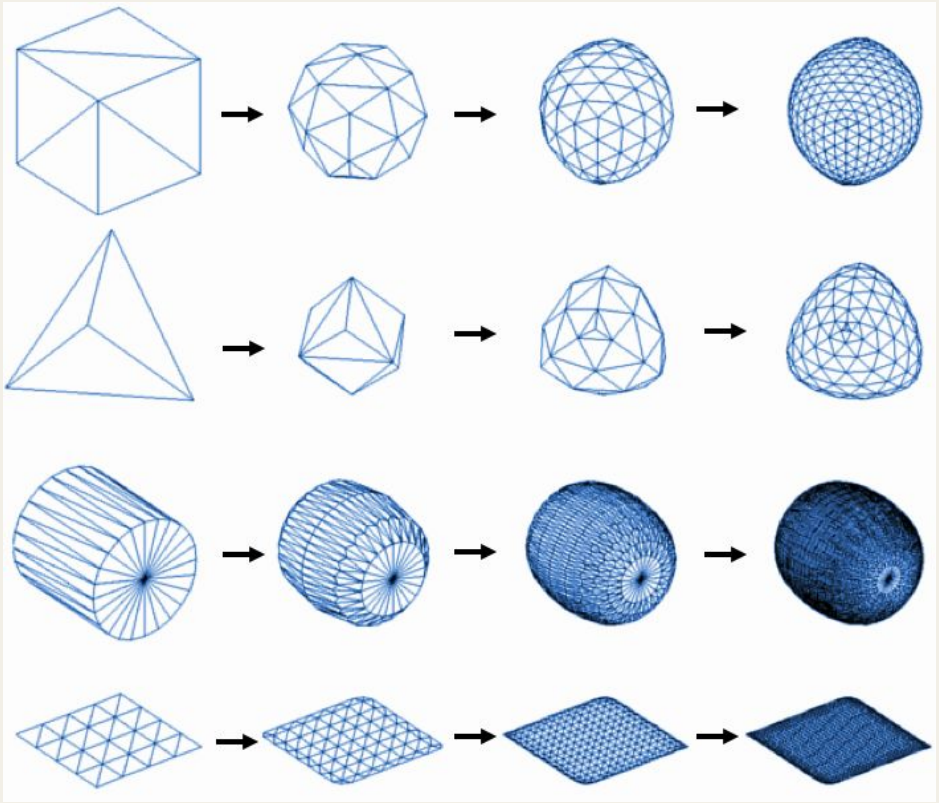


Figure 2.1: *Example of subdivision for curves in the plane. On the left 4 points connected with straight line segments. To the right of it a refined version: 3 new points have been inserted "inbetween" the old points and again a piecewise linear curve connecting them is drawn. After two more steps of subdivision the curve starts to become rather smooth.*

Subdivision of Surfaces



Charles Loop

Smooth Subdivision Surfaces Based on Triangles

by

Charles Teorell Loop

A thesis submitted to the faculty of
the University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mathematics

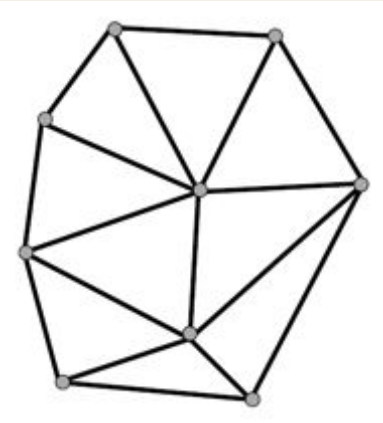
The University of Utah

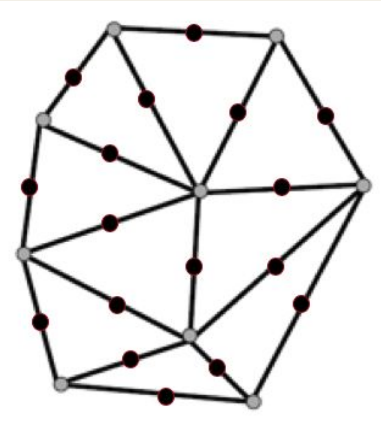
August 1987

2478 citations (and counting) MS thesis!

Loop Subdivision

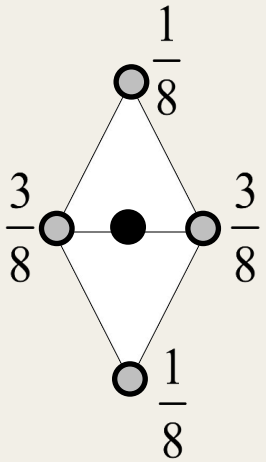
- Subdivide each triangle into 4 sub-triangles
- Move both the old/new vertices
- Repeat (if desired)
- C^2 continuity almost everywhere
  (except at some extraordinary vertices where it's only C^1)

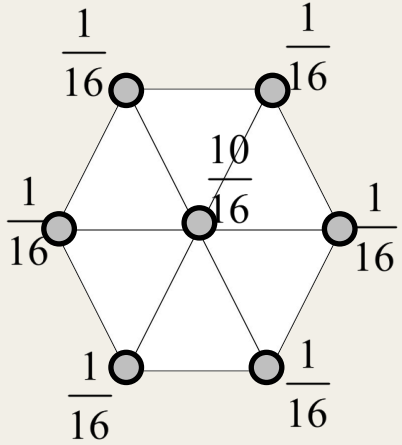




Move Old/New Vertices

- Perturb the position of each new vertex (black) using a weighted average of the four nearby original vertices (grey)
- Perturb the position of each regular original vertex (grey) using a weighted average of the six adjacent original vertices (grey)

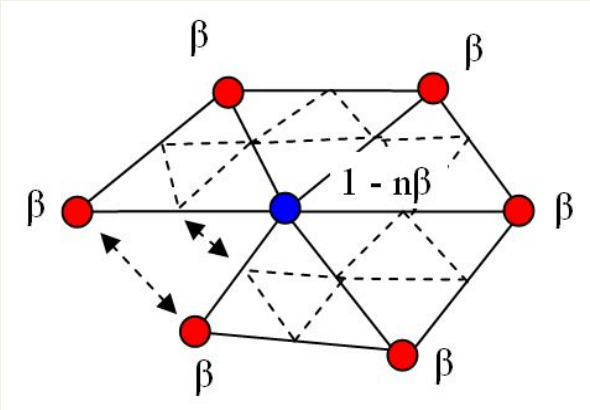




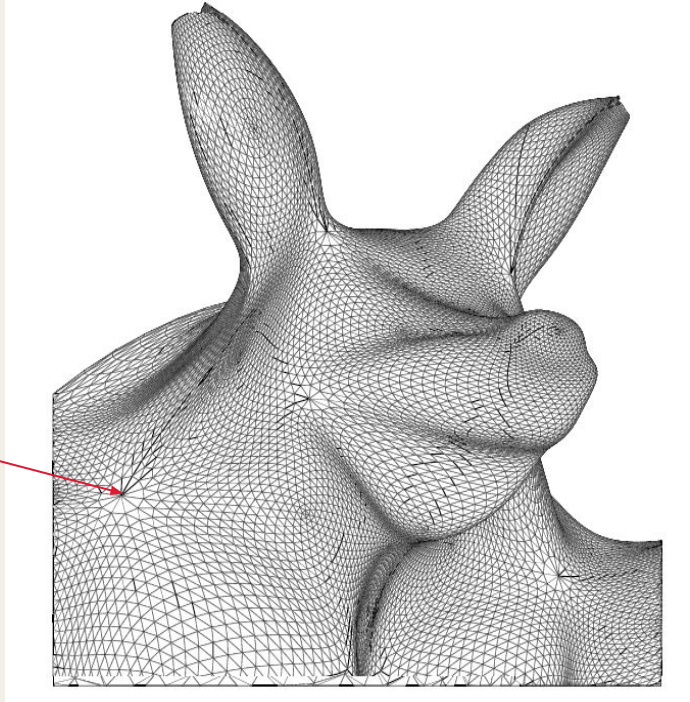
Extraordinary Points

- Most vertices are regular (degree 6)
- At extraordinary points, we can use Warren weights:

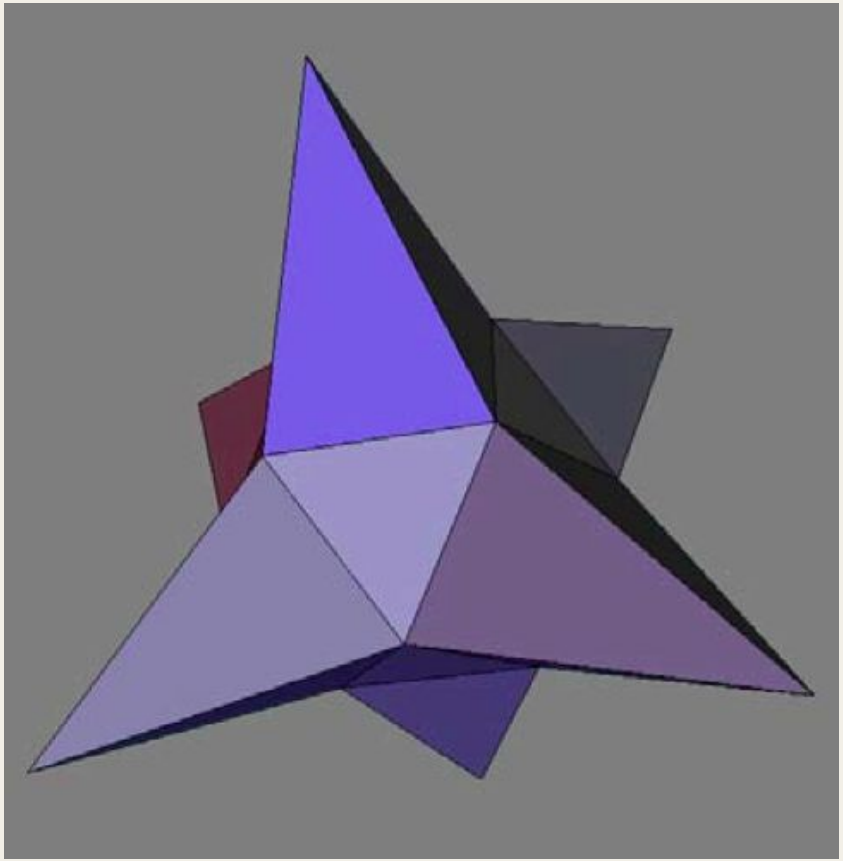
$$\beta = \frac{3}{16}, n = 3 \qquad \beta = \frac{3}{8n}, n \neq 3$$



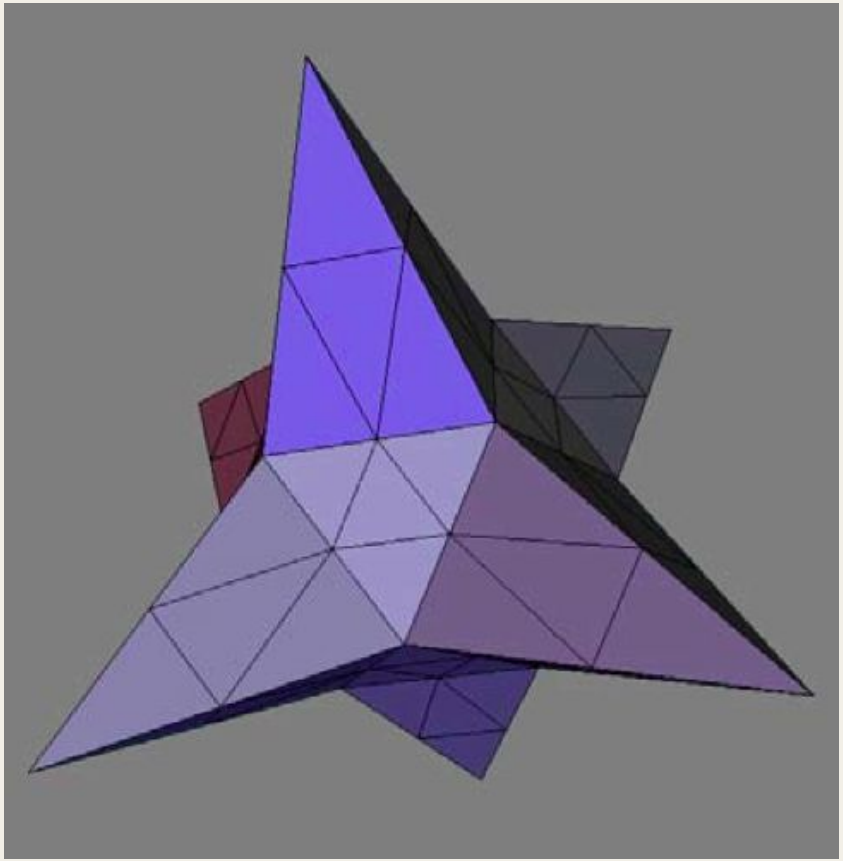
extraordinary point



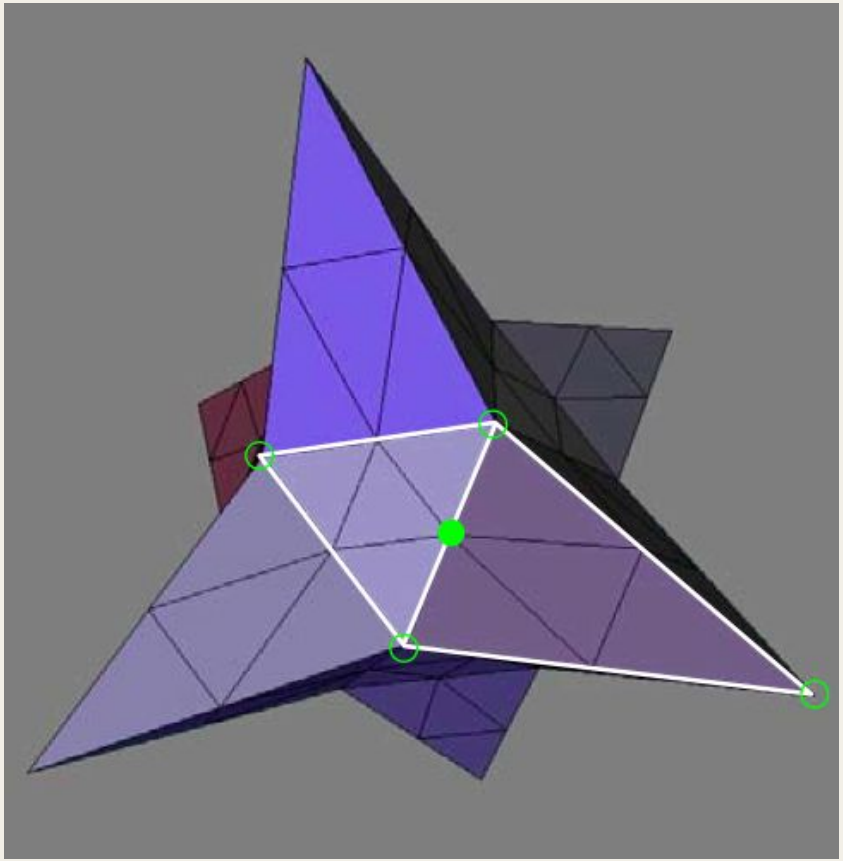
Example: Initial Mesh

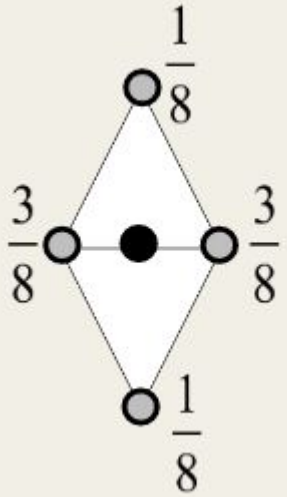


Add New Vertices

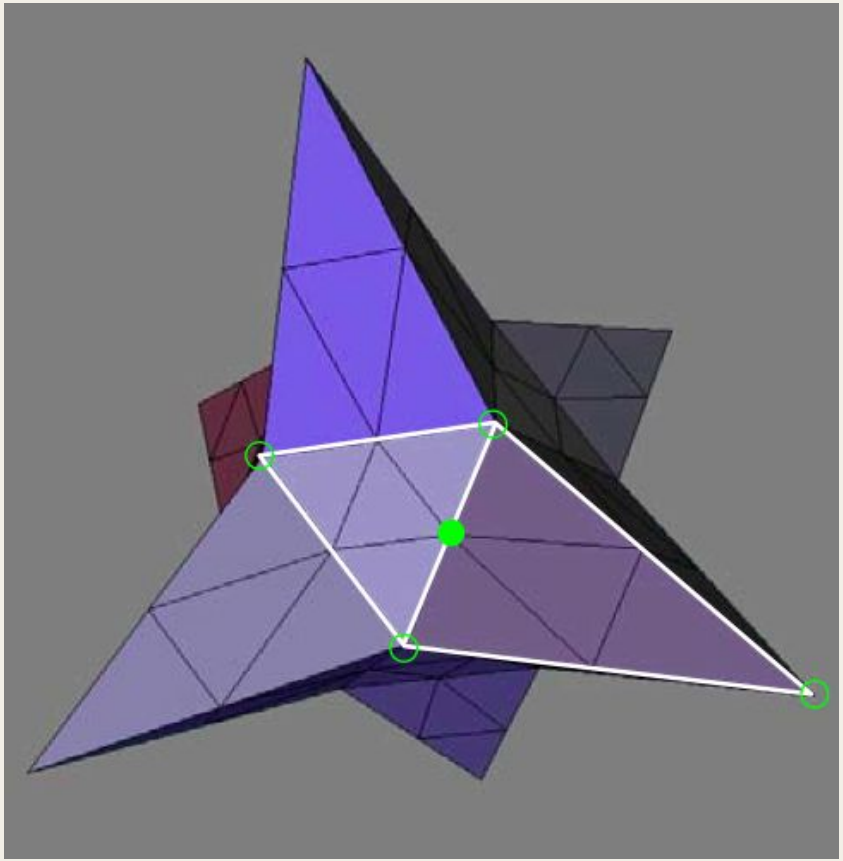


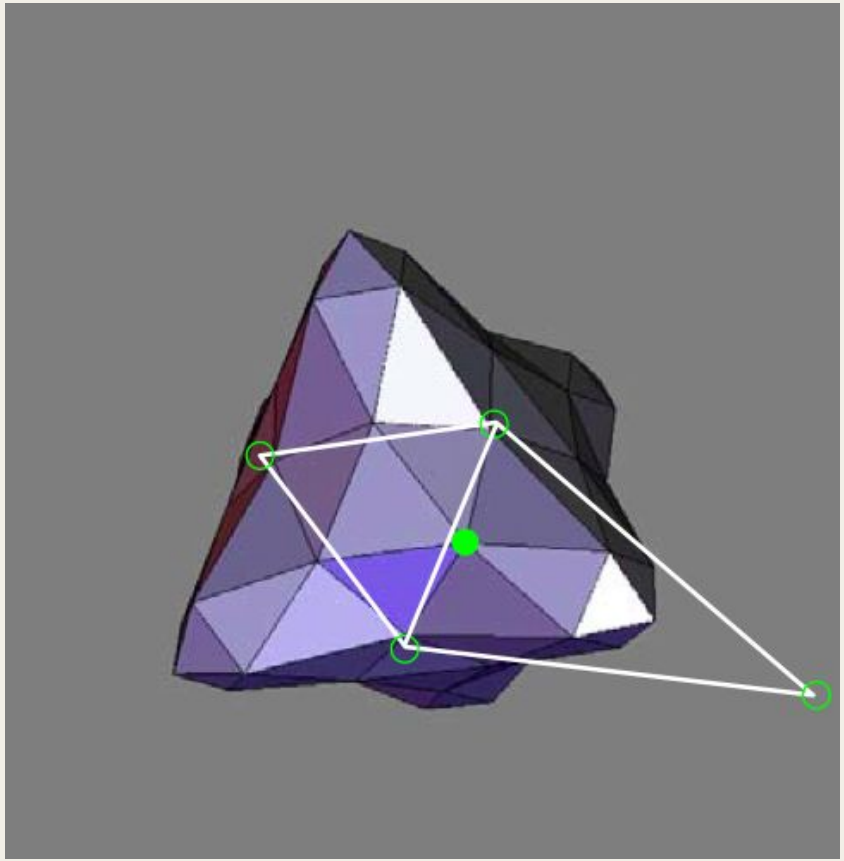
Moving New Vertices



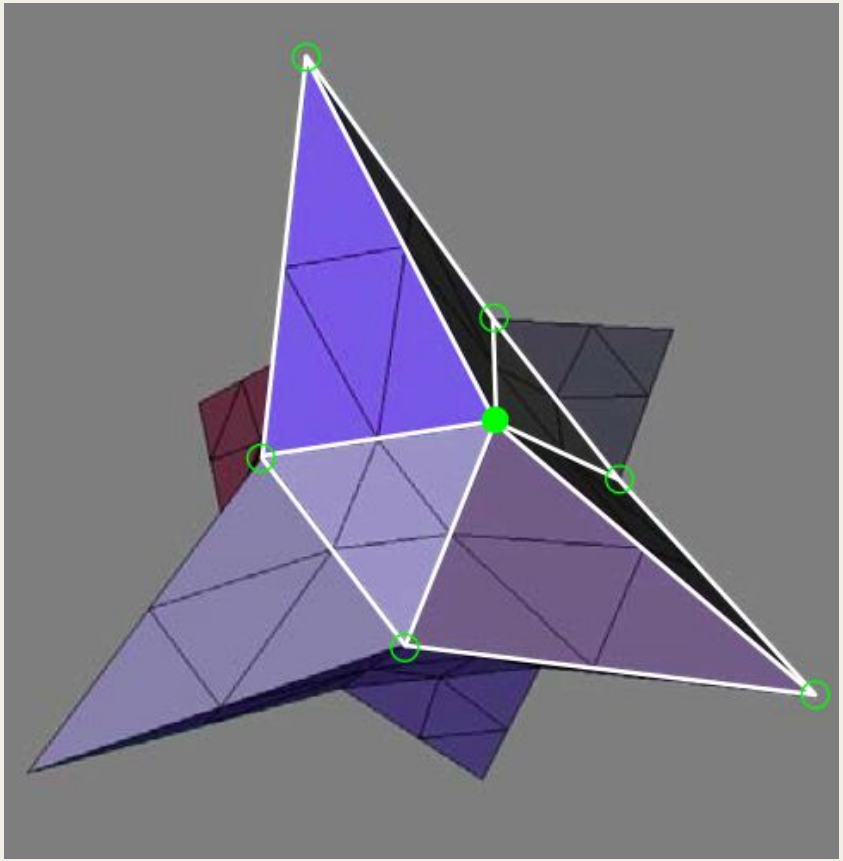


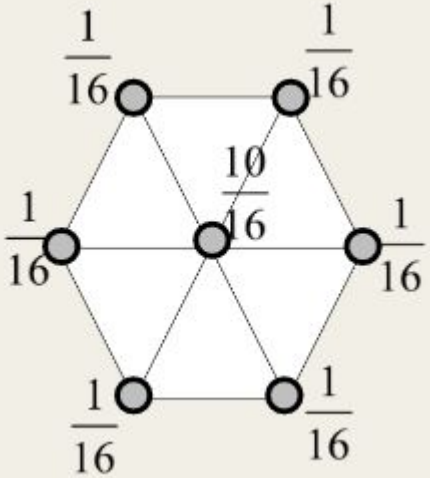
Moving New Vertices



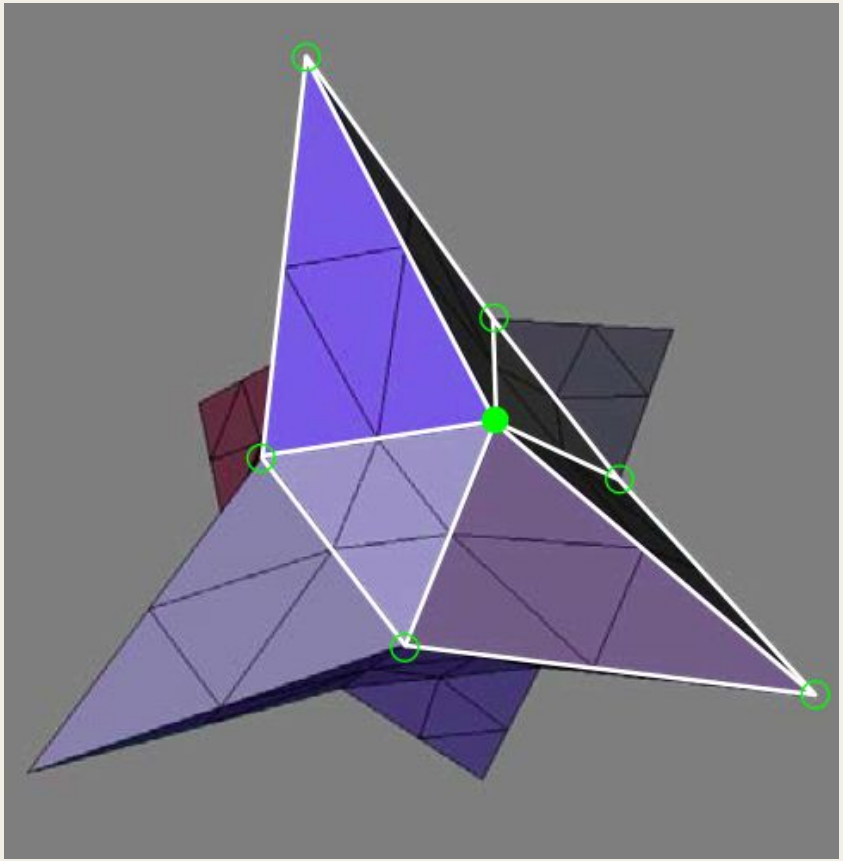


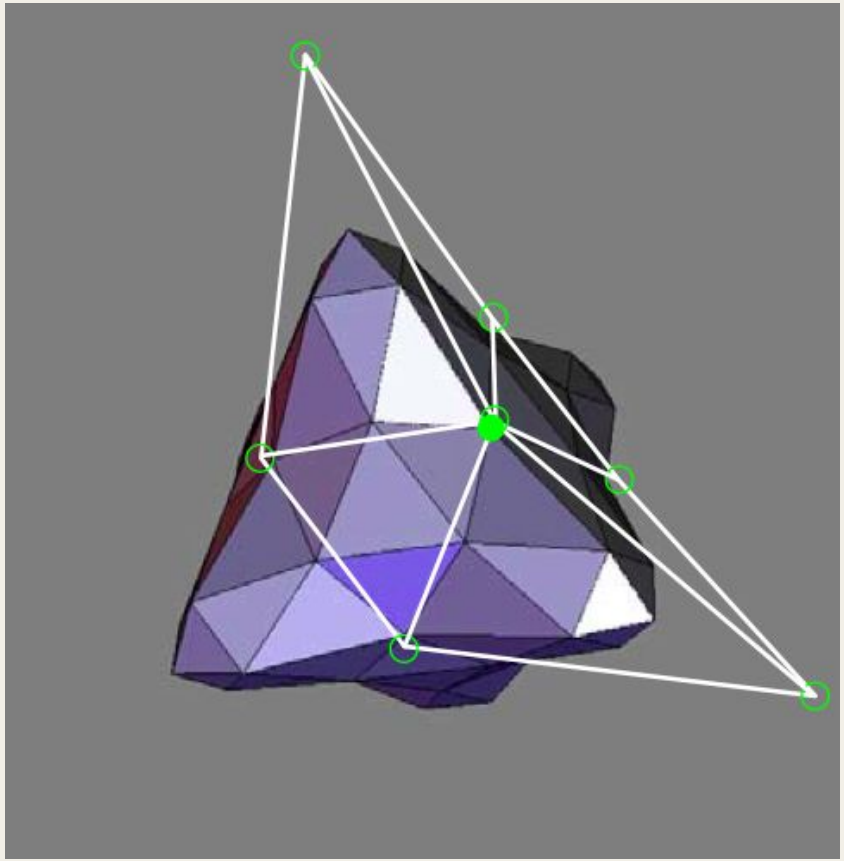
Moving Old Vertices



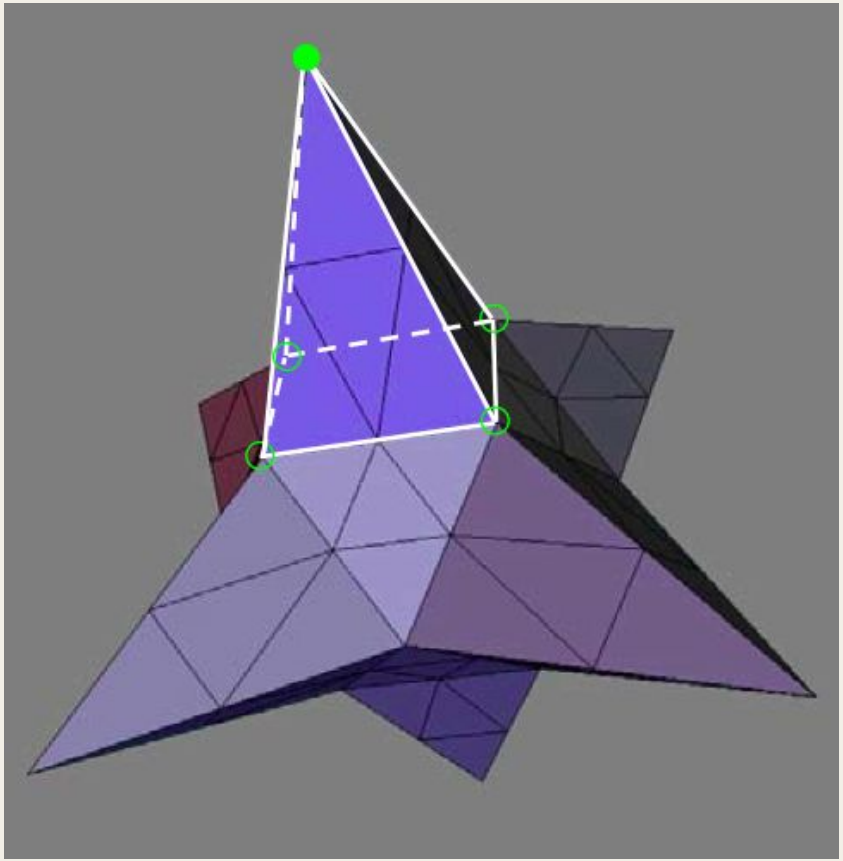


Moving Old Vertices

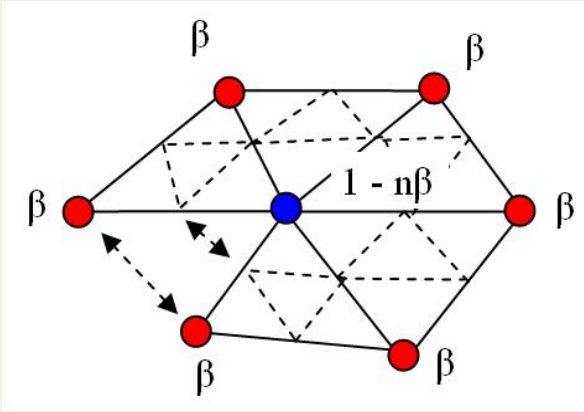




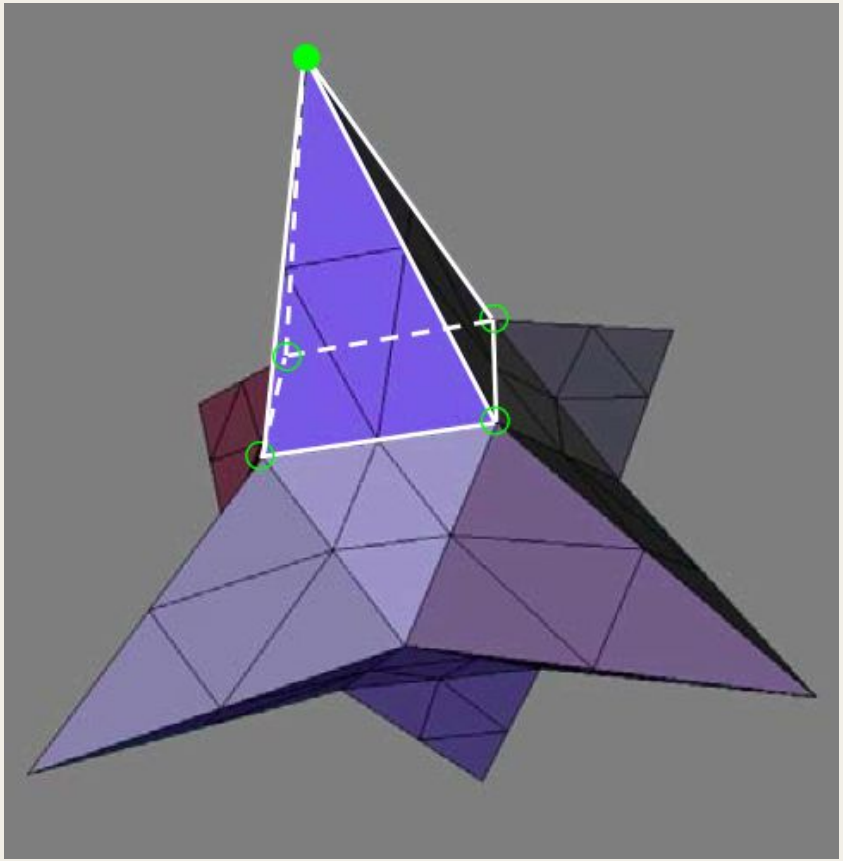
Extraordinary Vertices

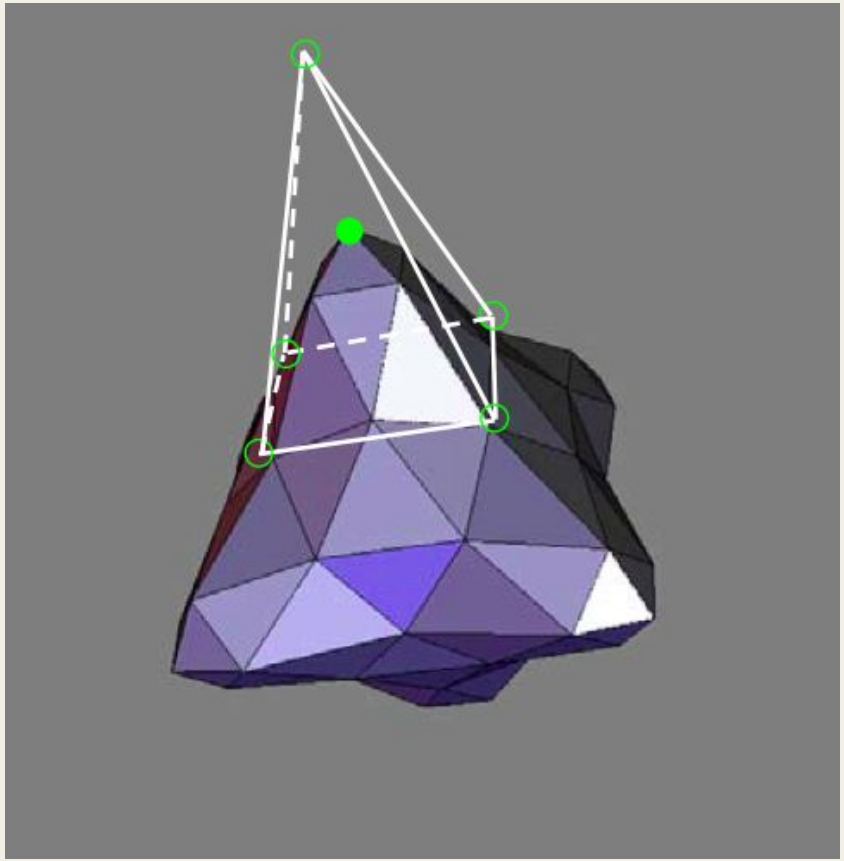


$$\beta = \frac{3}{16}, n = 3 \qquad \beta = \frac{3}{8n}, n \neq 3$$



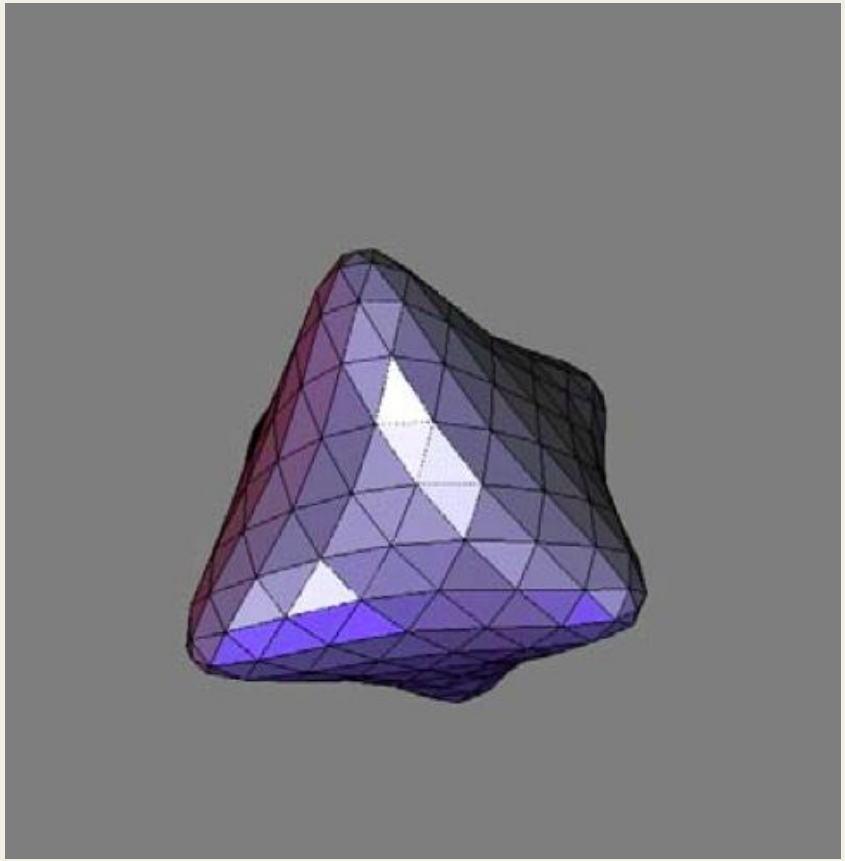
Extraordinary Vertices



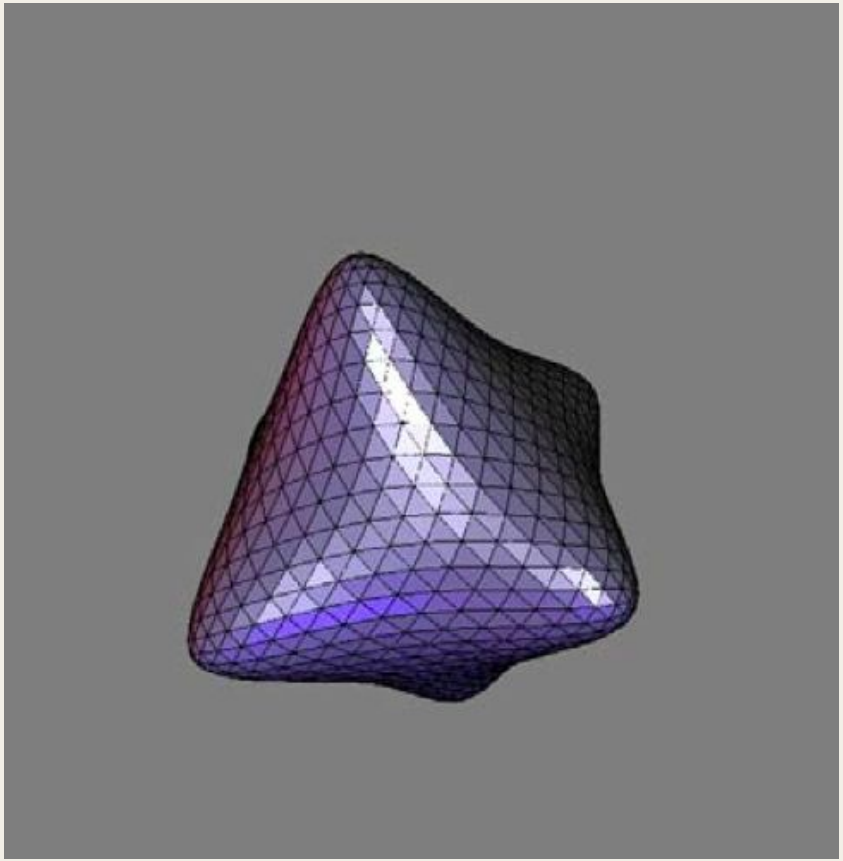


Result from One Subdivision

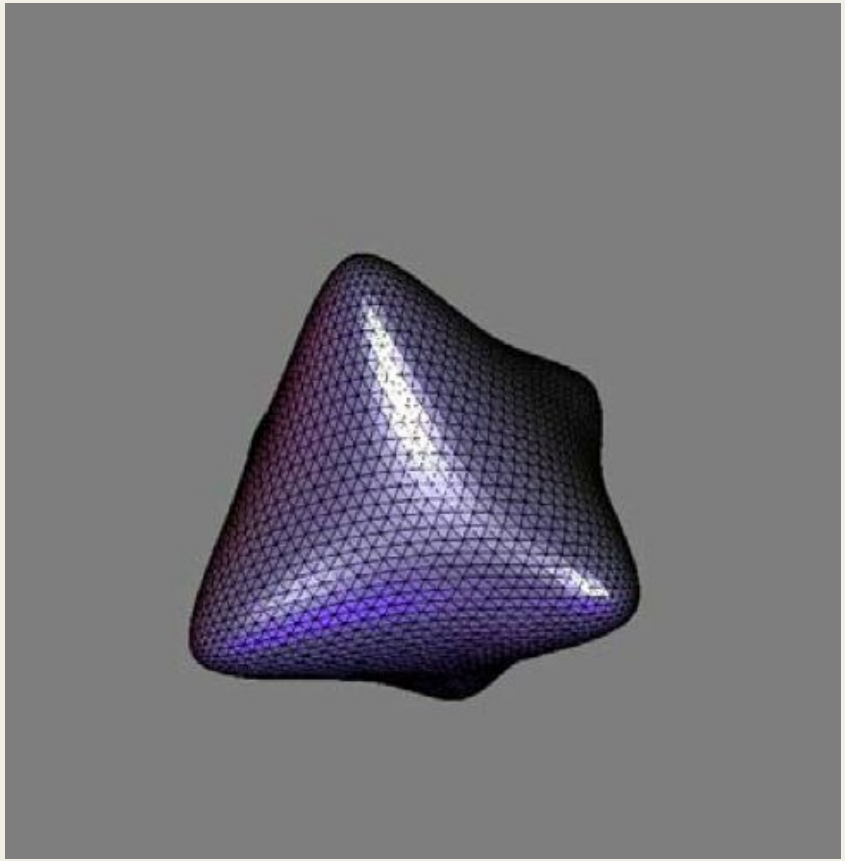
Two Subdivisions



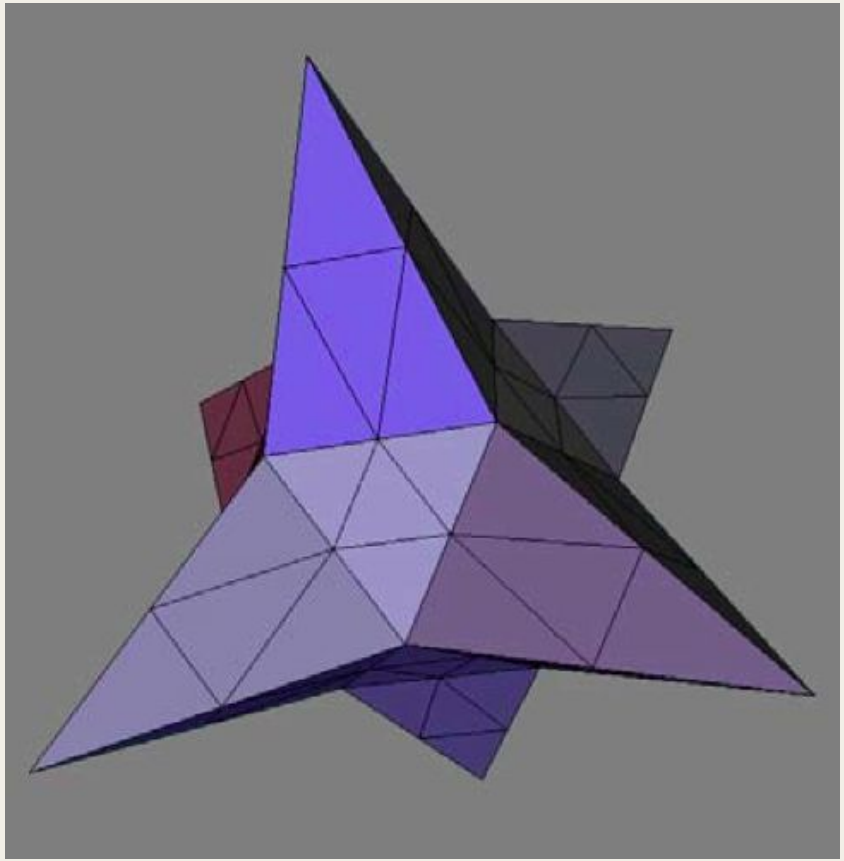
Three Subdivisions

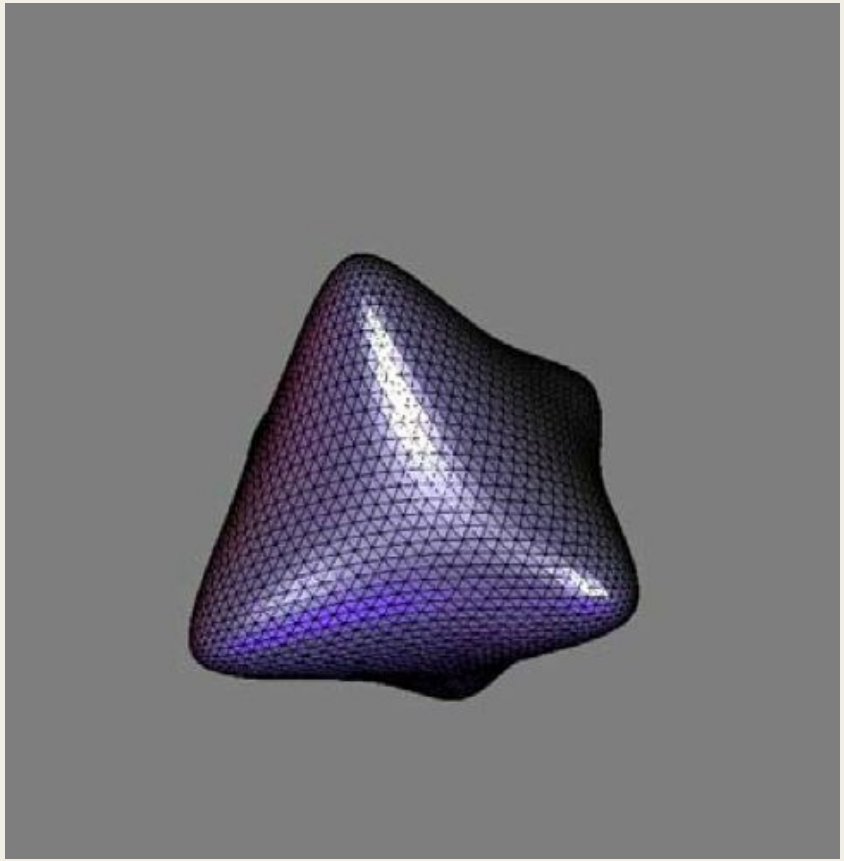


Four Subdivisions



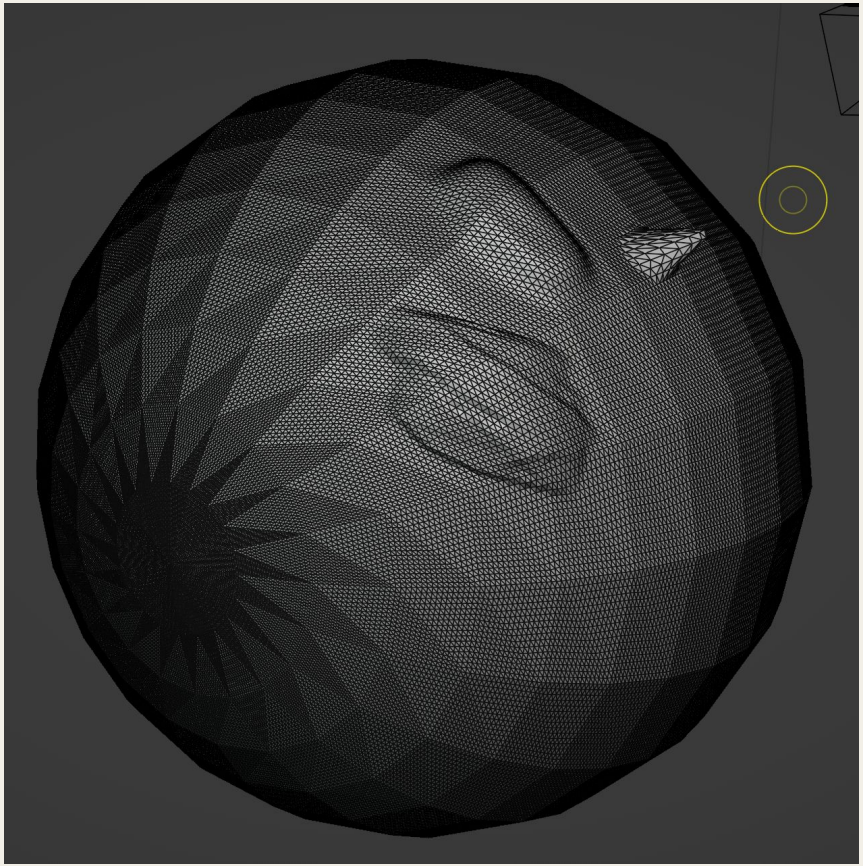
Loop Subdivision

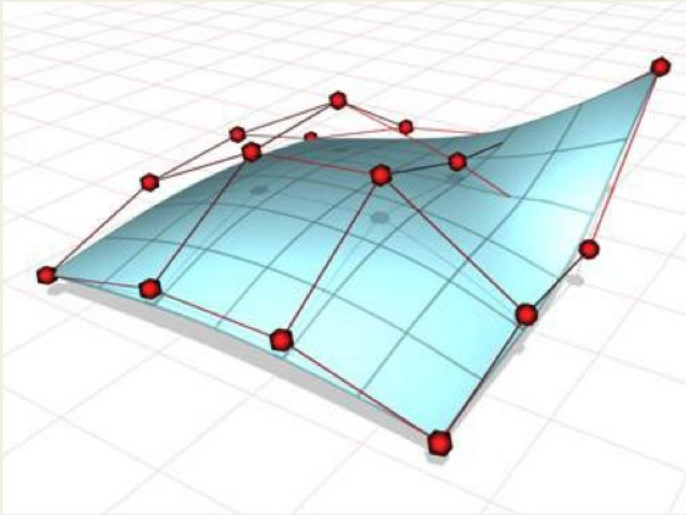




Questions?

Motivation: Deforming a Mesh

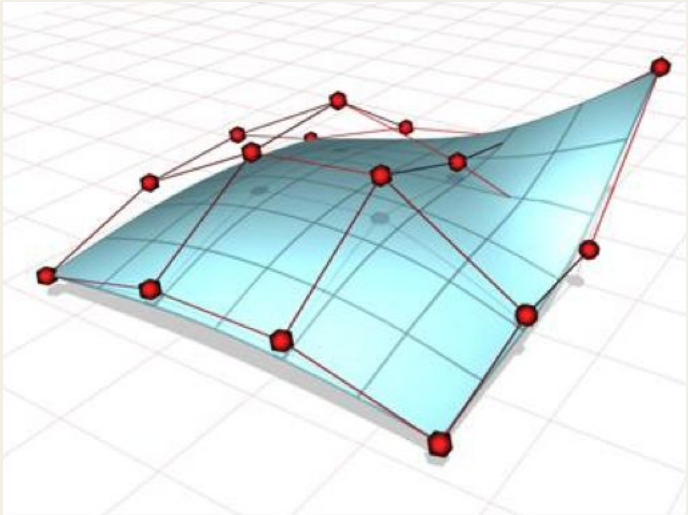




Splines

- Moving/placing every individual vertex when creating the initial geometry can be too time-intensive
- Would rather move just one vertex and have the rest move smoothly along

- Can be done with spline interpolation
- Essentially what happens when you deform/sculpt objects in Blender



Remember Linear Interpolation

- Suppose we have some important quantities at two points p_0 and p_1, and we want to interpolate the quantity inbetween:

$$f(u) = (1-u)p_0 + up_1$$

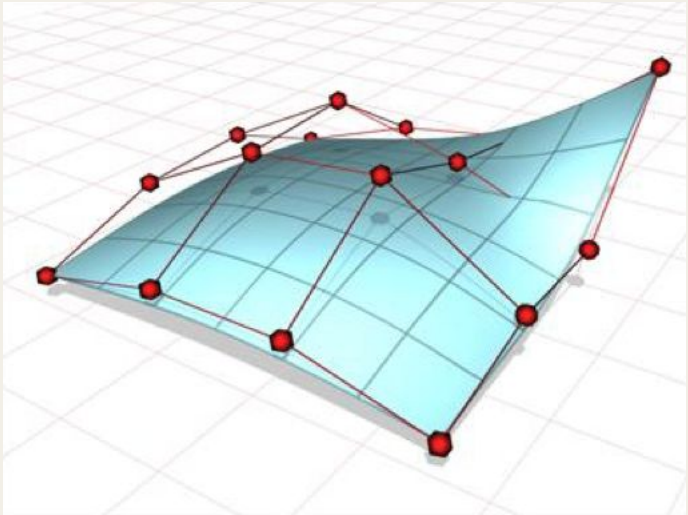
  where our parameter u is between [0, 1]

- Expressing this as a polynomial: $f(u) = a_0 + ua_1$
  where $f(0) = p_0$ and $f(1) = p_1$ and the a's can be solved:

$$\begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

Linear Not Smooth Enough

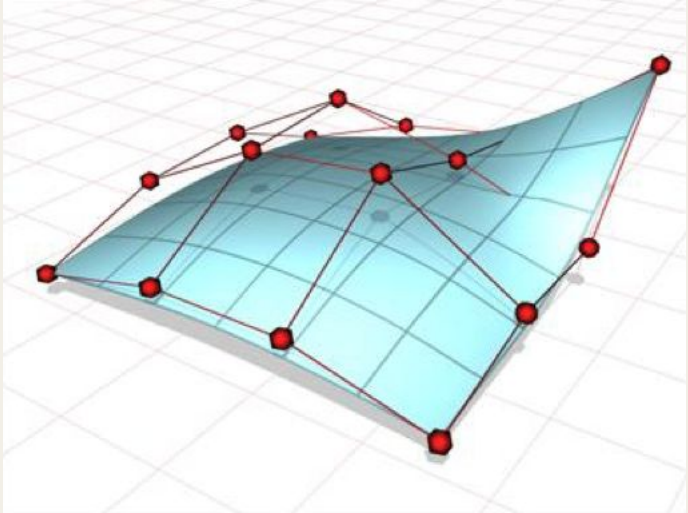
- Given a bunch of control points, we want to smoothly move the rest
- But if we only do linear interpolation, then we get sharp movement
- We want at least C^1 continuity



Linear Not Smooth Enough

- Given a bunch of control points, we want to smoothly move the rest
- But if we only do linear interpolation, then we get sharp movement
- We want at least C^1 continuity

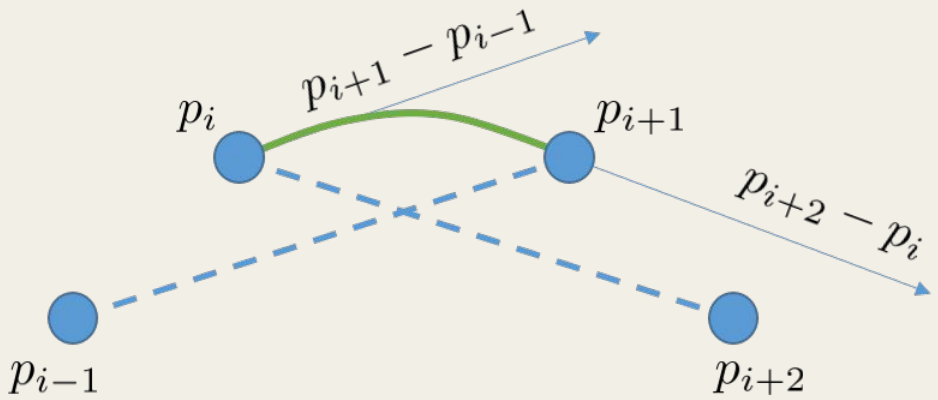
- Turns out cubics are enough for most applications, since C^1 & C^2 continuity
- Many kinds of cubic splines



Cardinal Cubic Splines

- 4 control points
- Derivative at the 2nd control point connects the 1st & 3rd
- Derivative at the 3rd control point connects the 2nd & 4th

- Construction of derivatives makes 2 consecutive curves continuous!



Cardinal Cubic Splines

- Interpolate 2nd & 3rd points
- Similar to what we did with linear interpolation, write the cubic polynomial:

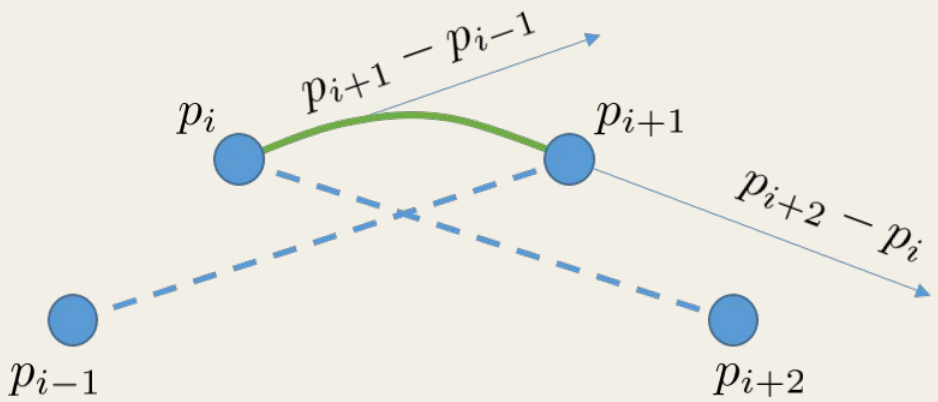
$$f(u) = a_0 + ua_1 + u^2a_2 + u^3a_3$$

$$f(0) = p_i$$

$$f(1) = p_{i+1}$$

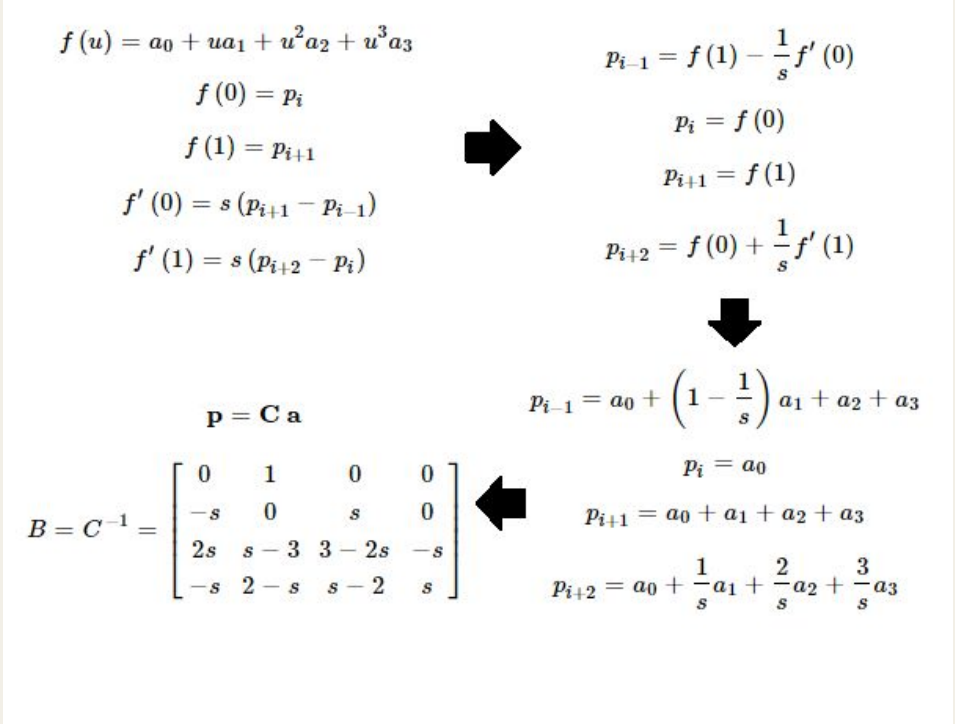
$$f'(0) = s(p_{i+1} - p_{i-1})$$

$$f'(1) = s(p_{i+2} - p_i)$$

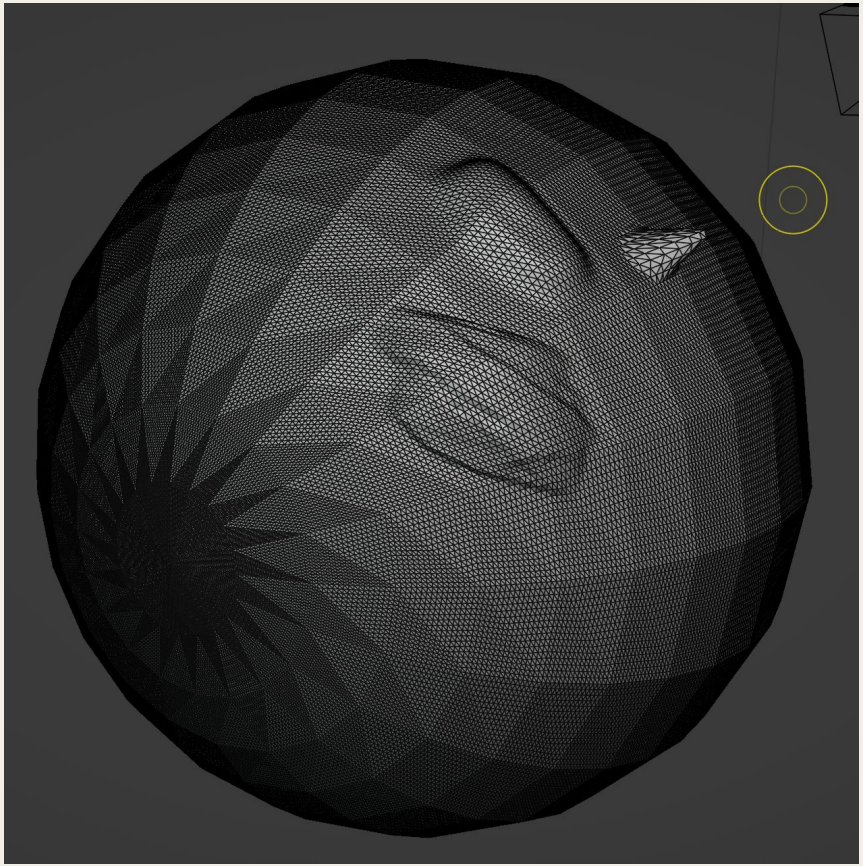


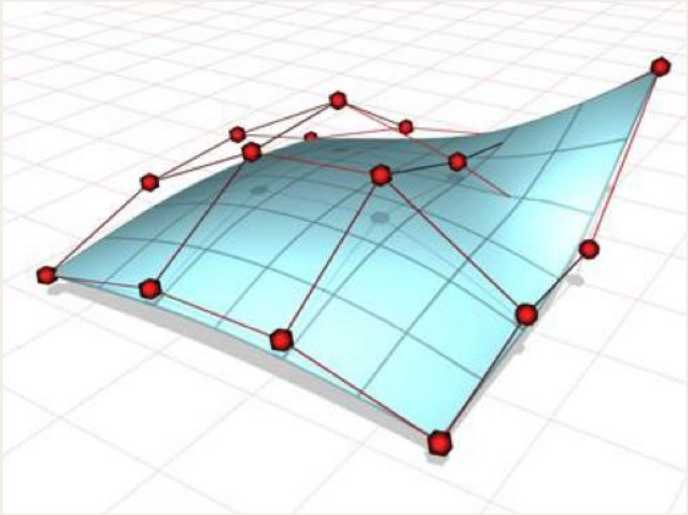
Cardinal Cubic Splines

- For those interested in the full derivation –
- We call the derived matrix the basis matrix
- The inverse is the called the constraint matrix
- Cardinal splines only one type of cubic splines
- Also B-splines & NURBS…



Interpolations are Local



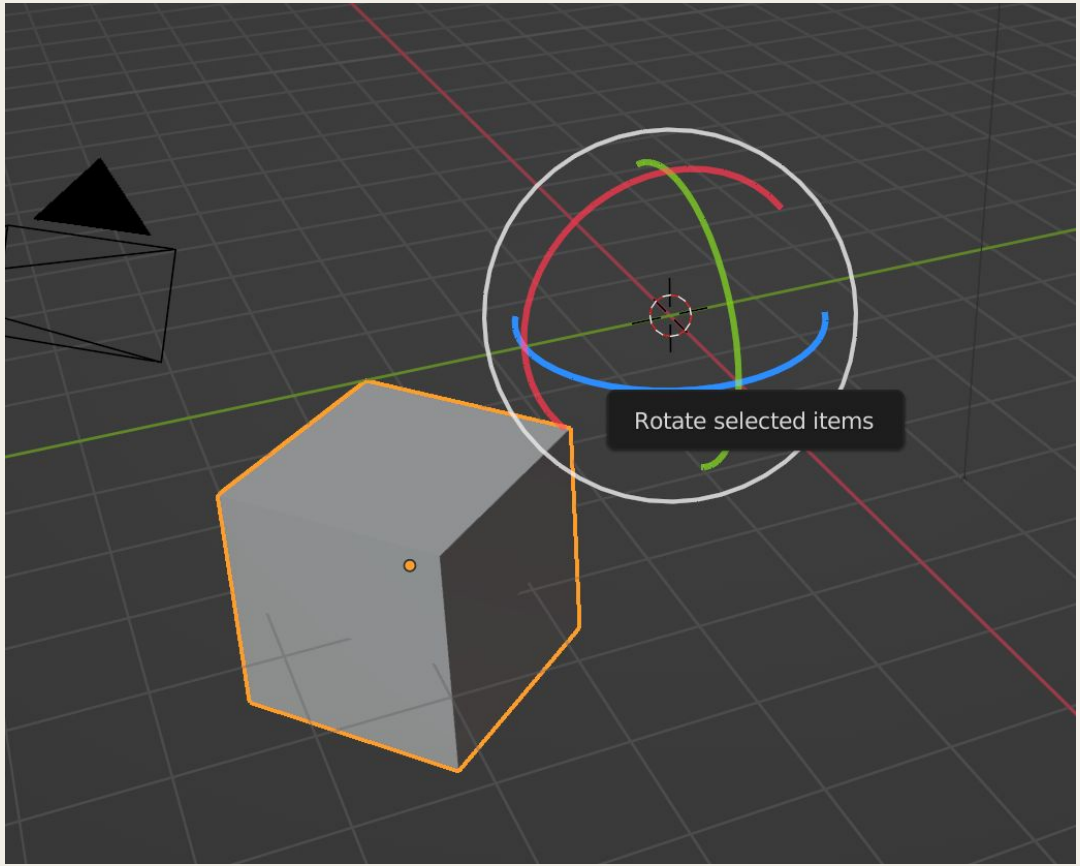


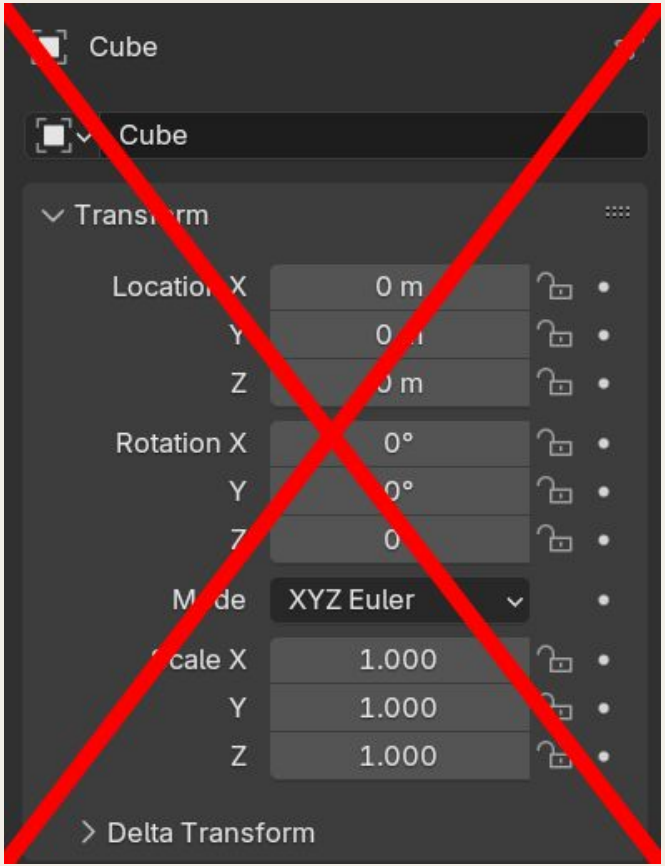
Questions?

Transforming Objects

- We create objects in a reference space called object space
- After creating them, we place the objects into the scene, which we refer to as world space
- Local vs. Global
- Placing objects may require: rotating, scaling (resizing), translating

Local vs. Global





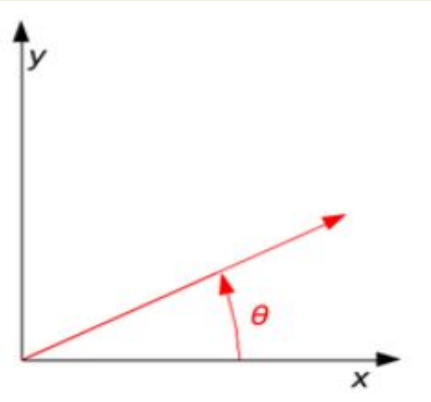
Transforming Objects

- We create objects in a reference space called object space
- After creating them, we place the objects into the scene, which we refer to as world space
- Local vs. Global
- Placing objects may require: rotating, scaling (resizing), translating

- First consider a single 3D point with (x, y, z) coordinates
- An object is just a bunch of points or vertices, so handling 1 point extends to handling the whole object

Rotation

- In 2D, we rotate a point counter-clockwise about the origin of a Cartesian coordinate space via:



$$\begin{pmatrix} x^{new} \\ y^{new} \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = R(\theta) \begin{pmatrix} x \\ y \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation

- For 3D, we have the rotation matrices for about the x, y, z axes respectively:

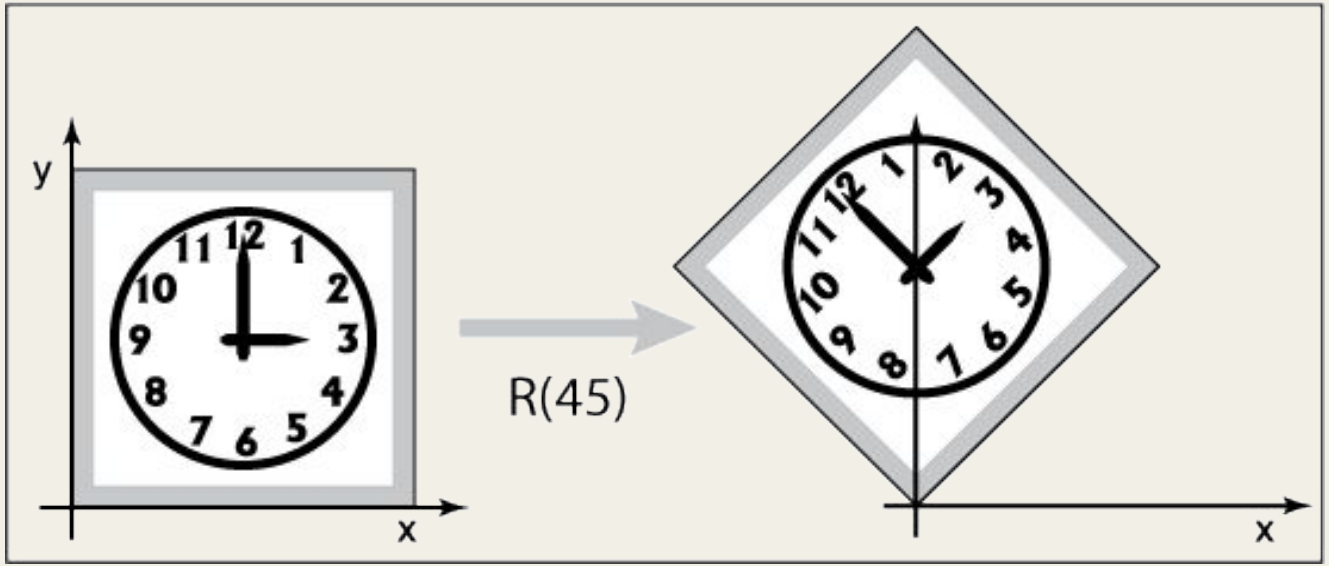
$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation

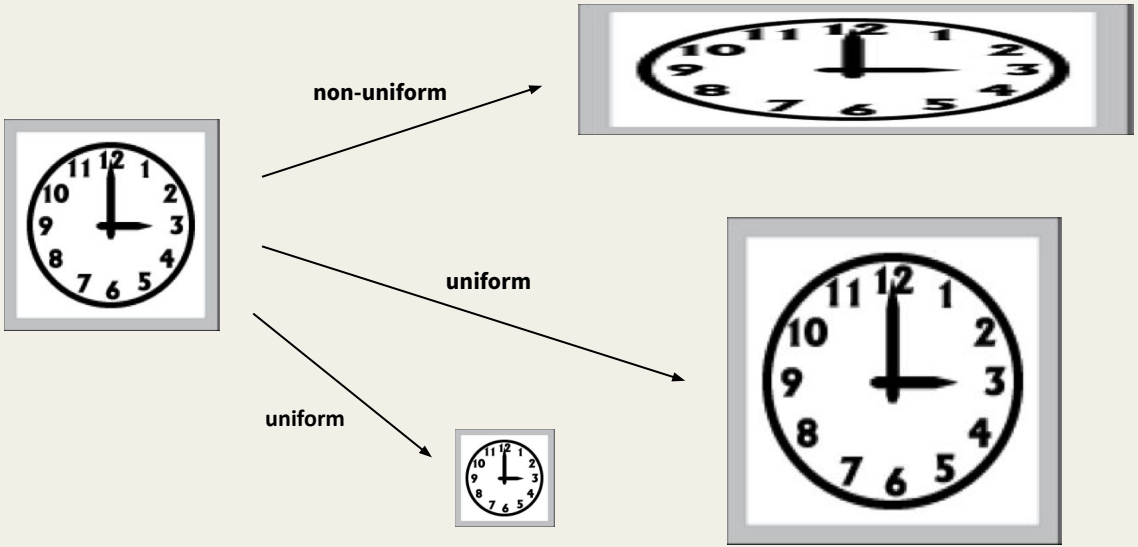
- Note that rotations preserve shape:



- Order of rotation matters! Matrix multiplication is not commutative!

Scaling (aka Resizing)

- A scaling matrix has the form: $S = \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{pmatrix}$
- Scaling matrices can both resize and shear/stretch objects:



Homogeneous Coordinates

- To represent translation with matrices, we need to use homogeneous coordinates

- In general, the homogeneous coordinates of a point in 3D are:

$$\vec{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \vec{p}_H = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}, w \neq 0$$

- Let the 4th component be 1, so we have: $\vec{p}_H = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$

Translation

- To translate a point some amount $\vec{t} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$ we use the 4x4 matrix:

$$\begin{pmatrix} & I_{3x3} & & t_1 \\ & & & t_2 \\ & & & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \vec{x} + \vec{t} \\ 1 \end{pmatrix}$$ with the identity: $I_{3x3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

- Intuitively, this should confirm with what you'd expect with translation:
  - moving a point by some vector simply adds that vector to the point to get the new location

Transforming in Homogeneous Coordinates

- Rotation and scaling matrices can also be expressed in homogeneous coordinates as:

$$\begin{pmatrix} & M_{3x3} & \\ & & \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} M_{3x3}\vec{x} \\ 1 \end{pmatrix}$$

- Notice how we can simply take off the fourth component, the 1, once we're done with our transformations and get our desired transformed vertex

Transforming in Homogeneous Coordinates

- In short, to transform a 3D object:
  - Convert all vertices in the object to homogeneous coordinates by simply making them 4D vectors with a fourth 1 component
  - Apply your transformation matrices for translation, rotation, scaling by left-multiplying each vertex in the order you want to apply the transforms
  - Convert all vertices back to 3D by taking off the fourth component once you're done

Questions?