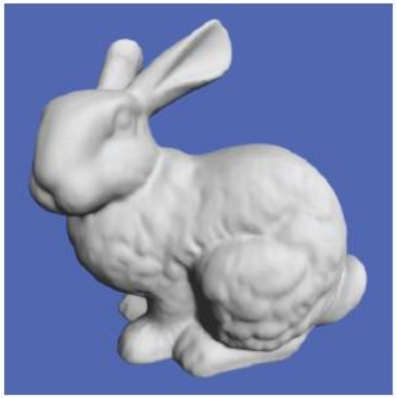
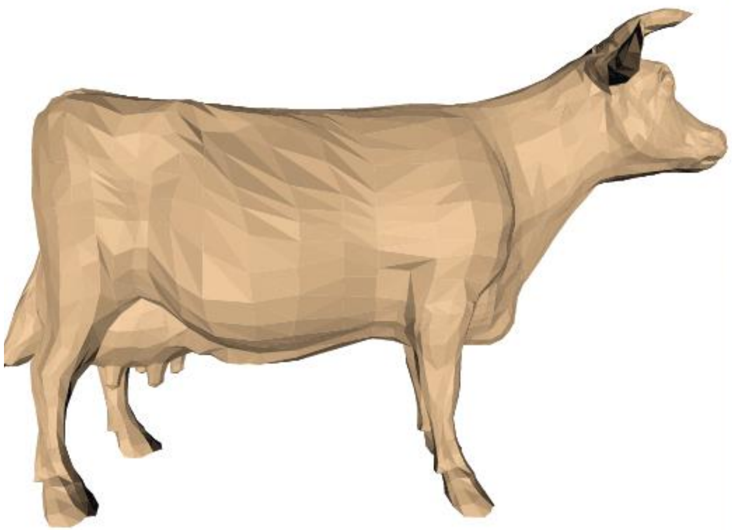
CSE160 – Oct 8

- Everything is triangles
- OpenGL Primitives
- How is this stored in buffers
- Rasterization
- Normals
- Interpolation
- Non triangle modeling
- Assignment 1
- Administrative
- Q&A

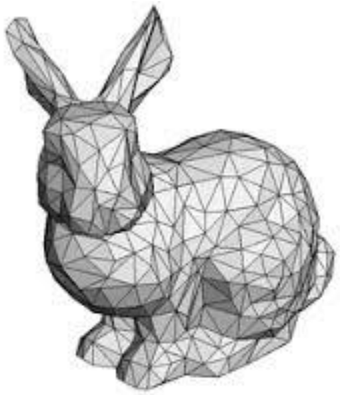
Everything is made of ~~atoms~~ triangles

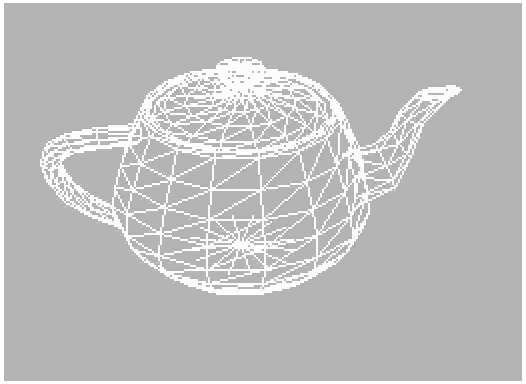
Triangle Meshes

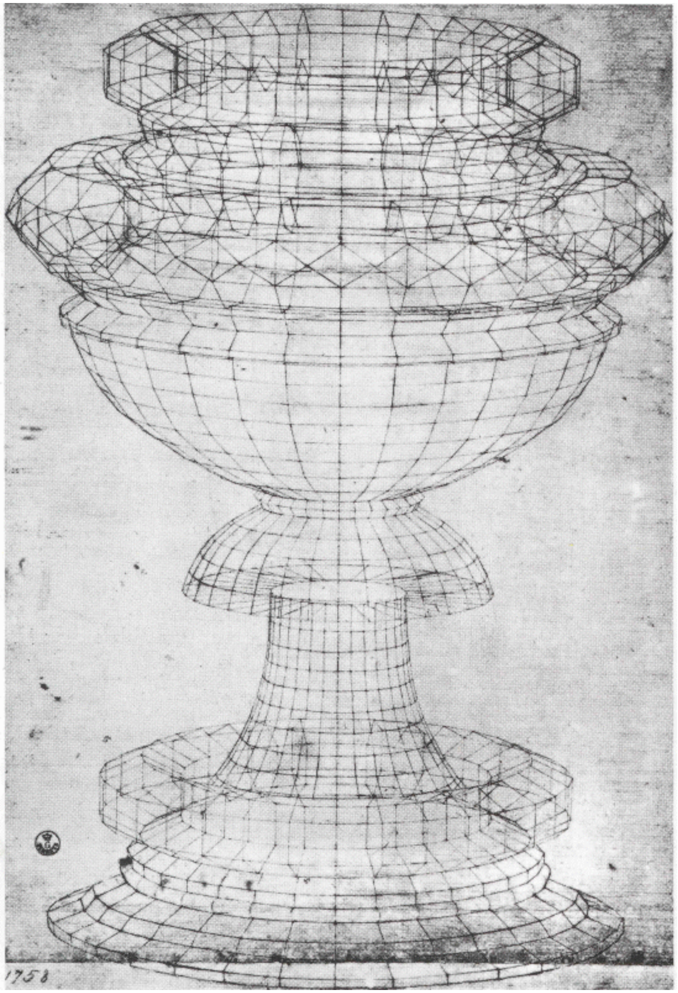




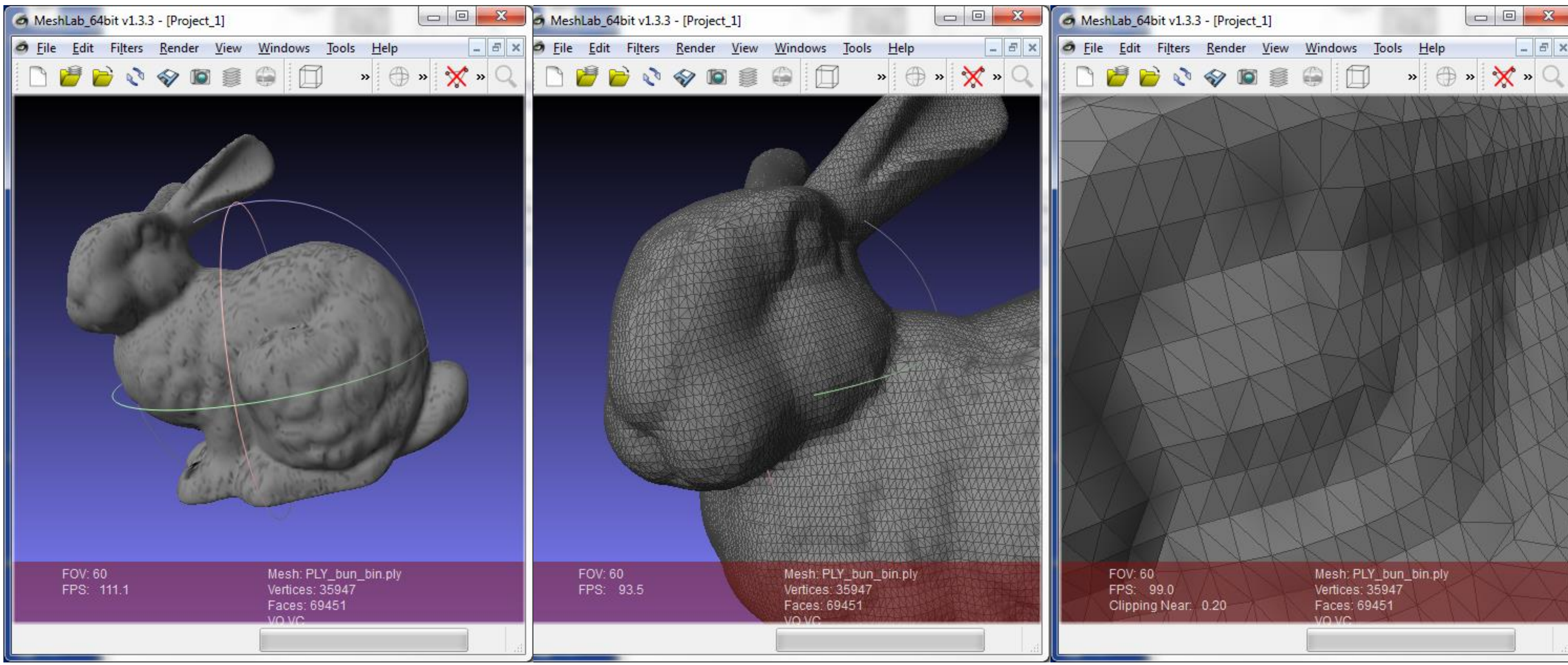
Everything made of triangles







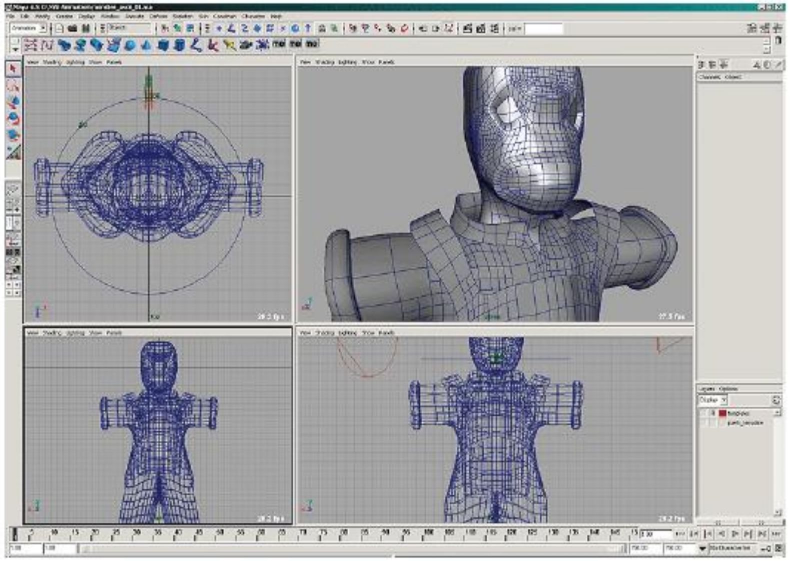
Pen and ink drawing of a wireframe chalice ("Perspective Study of a Chalice"), done by Paolo Uccelloin 1430-1440, Florence, Italy.


MeshLab_64bit v1.3.3 - [Project_1]
File Edit Filters Render View Windows Tools Help
FOV: 60
FPS: 111.1
Mesh: PLY_bun_bin.ply
Vertices: 35947
Faces: 69451
MeshLab_64bit v1.3.3 - [Project_1]
File Edit Filters Render View Windows Tools Help
FOV: 60
FPS: 93.5
Mesh: PLY_bun_bin.ply
Vertices: 35947
Faces: 69451
MeshLab_64bit v1.3.3 - [Project_1]
File Edit Filters Render View Windows Tools Help
FOV: 60
FPS: 99.0
Clipping Near: 0.20
Mesh: PLY_bun_bin.ply
Vertices: 35947
Faces: 69451

Mesh Generation

Modeling

- Software packages like Maya, Blender, etc. are powerful but hard to use
- Tremendous time investment needed to create complex models



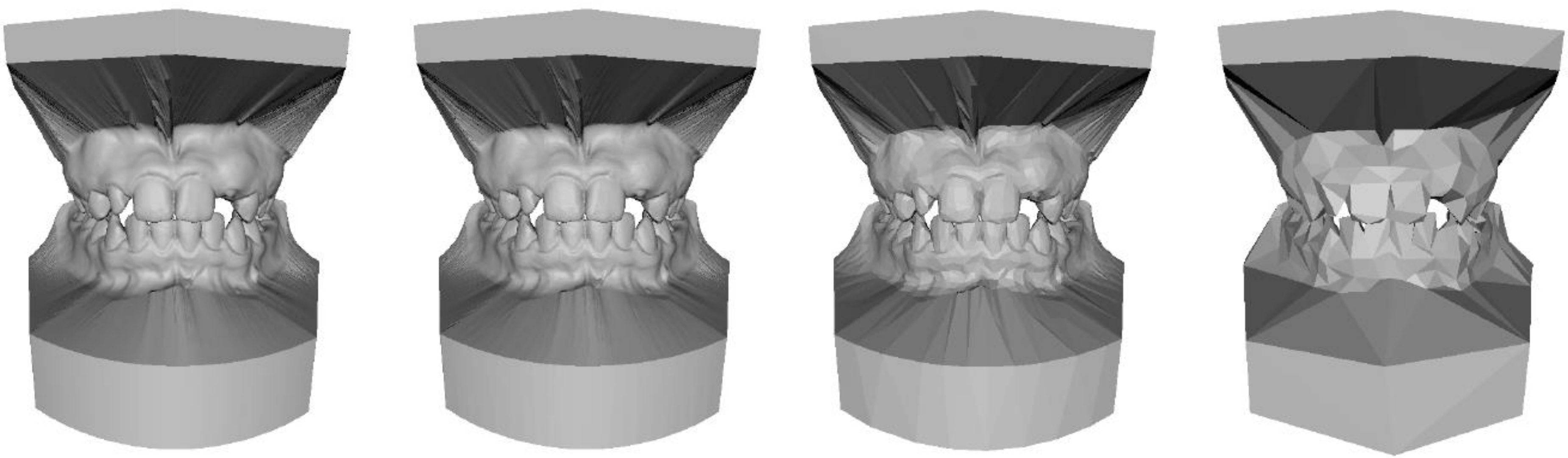
Laser Scanning

- Good for capturing real objects
- Scanners are expensive
- Registering multiple scans is difficult
- Turning point data into triangles is also non-trivial

Level of Detail

Far Away Objects Need Less Detail

- Acquisition systems often produce huge models
- Create multiple versions of models
- Pick the correct version for each view
- Can result in substantial performance gains
- Simplification is nontrivial



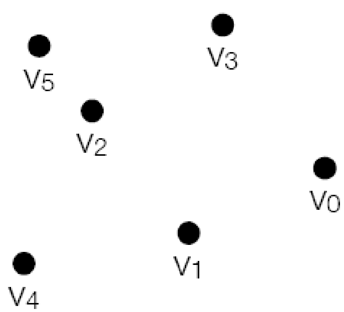
424,376 60,000 8,000 1,000

OpenGL Primitives

Points in OpenGL

GL_POINTS

- Draws square pixel region on screen
- One pixel wide by default
- With antialiasing, circular region drawn with smooth edges
- Size controllable with `glPointSize()`
- Easy, efficient way to activate pixels

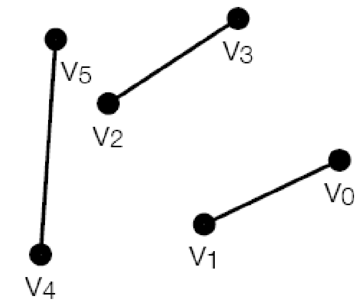



```
glPointSize(10.0f);
glBegin(GL_POINTS);
  for(int i=0; i<N; i++)
    glVertex3fv(v[i]);
glEnd();
```

Lines in OpenGL

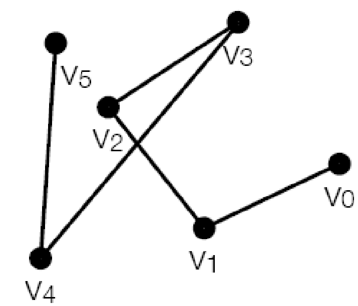
GL_LINES

- Draws lines one pixel wide
- Width can be controlled by `glLineWidth()`
- Successive *pairs* of vertices specify segments



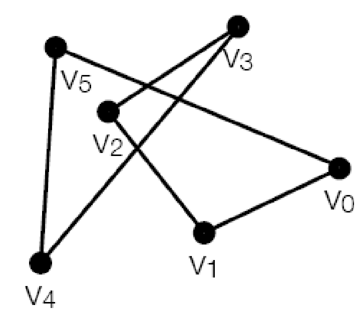
GL_LINE_STRIP

- Like GL_LINES, but successive vertices specify next connected segment in strip



GL_LINE_LOOP

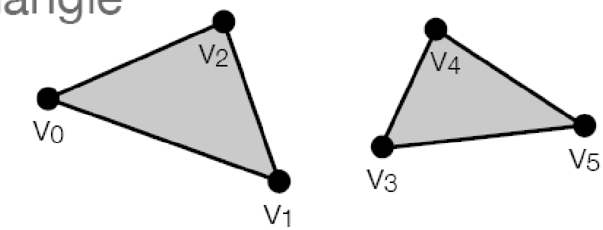
- Like GL_LINE_STRIP, but also connects last and first vertex



Triangles in OpenGL

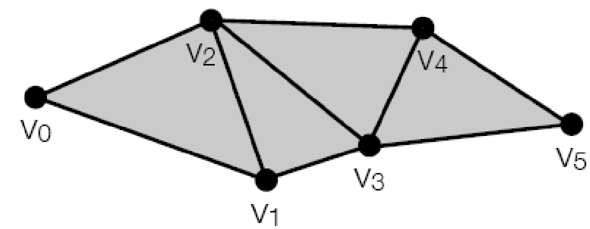
GL_TRIANGLES

- Successive vertex triples specify individual triangles
- Requires three vertices to be emitted for every triangle



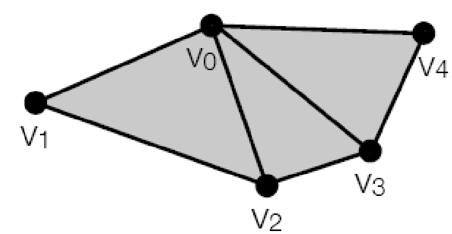
GL_TRIANGLE_STRIP

- First triple specifies first triangle
- Subsequent vertices *each* specify new triangle, along with previous two vertices
- One vertex emitted per triangle in long strips
- But stripifying meshes is nontrivial



GL_TRIANGLE_FAN

- First vertex is center of fan
- Subsequent vertices form ordered bounday
- One vertex emitted per triangle for dense fans
- But few such fans arise in practice

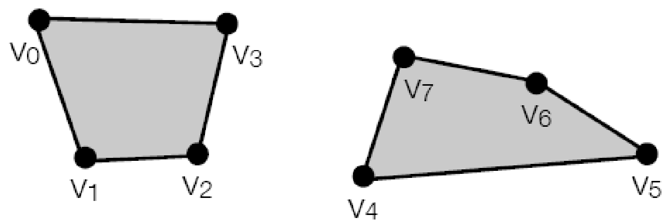


Other Primitives

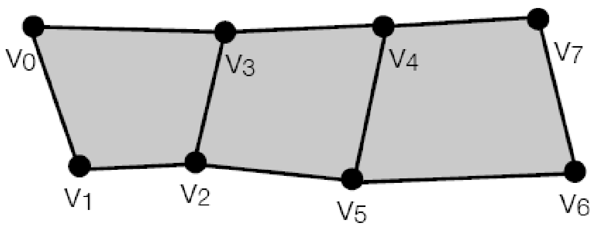
Not in WebGL

GL_QUADS

- OpenGL only handles **planar** quadrilaterals properly



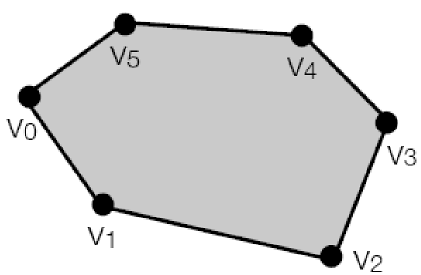
GL_QUAD_STRIP



These primitives are trouble.
It's safer to stick to triangles.

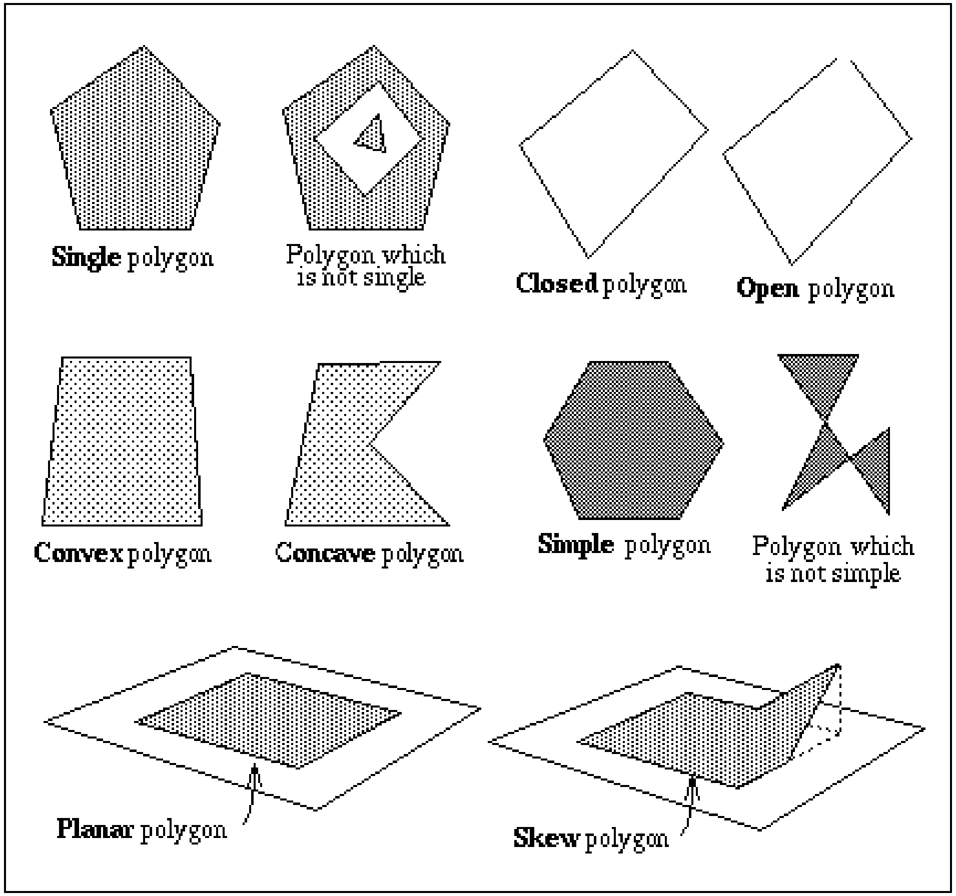
GL_POLYGON

- OpenGL only handles **convex** polygons properly

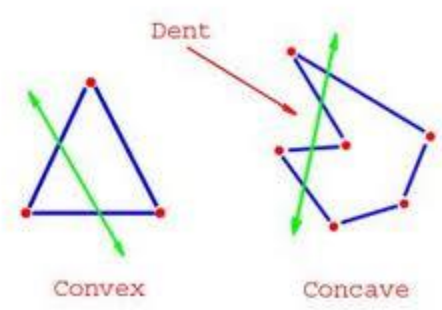


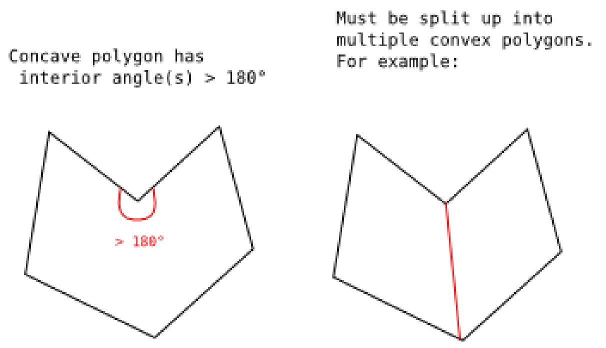
Reasons triangles are better

- Definitely planar
- Definitely convex
- Definitely not self intersecting
- Exactly 3 vertices always



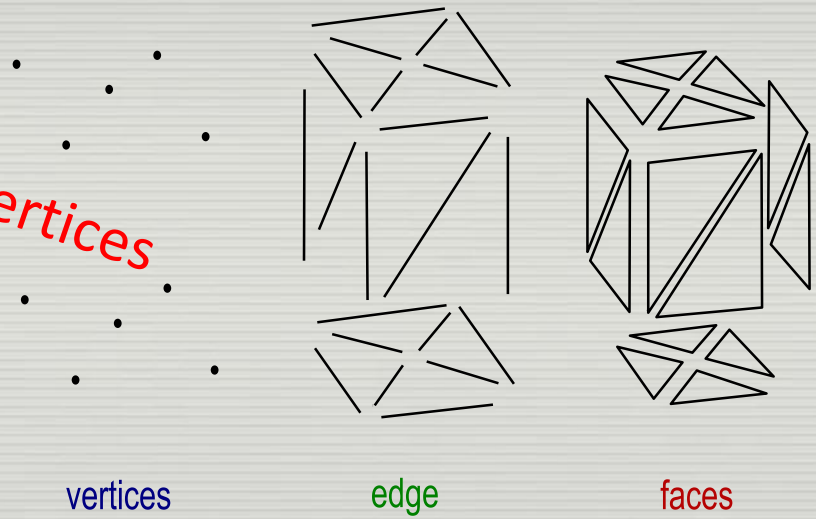
Testing convexity





Polygonal Meshes

You only specify vertices



(image courtesy of Wikipedia)

Q: What will render a square?

(A)
```
glBegin(GL_TRIANGLES)
  glVertex3f(0,0,0);
  glVertex3f(1,1,0);
  glVertex3f(1,0,0);
  glVertex3f(0,0,0);
  glVertex3f(0,1,0);
  glVertex3f(1,1,0);
glEnd();
```

(B)
```
glBegin(GL_QUADS)
  glVertex3f(0,0,0);
  glVertex3f(0,1,0);
  glVertex3f(1,0,0);
  glVertex3f(1,1,0);
glEnd();
```

(C)
```
glBegin(GL_TRIANGLES)
  glVertex3f(0,0,0);
  glVertex3f(1,1,0);
  glVertex3f(1,0,0);
  glVertex3f(0,0,0);
  glVertex3f(1,1,0);
  glVertex3f(0,1,0);
glEnd();
```

(D)
```
glBegin(GL_QUADS)
  glVertex3f(0,0,0);
  glVertex3f(0,1,0);
  glVertex3f(1,1,0);
  glVertex3f(1,0,0);
glEnd();
```

(E) I just really don’t know

How is this stored in buffers

[V0.x, V0.y, V1.x, V1.y, V2.x, V2.y, V3.x, V3.y, V4.x, V4.y, V5.x, V5.y, V6.x, V6.y, …]

Point 0 Point 1 Point 2

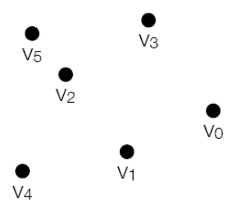
Attribute Vec2 a_Position;

drawArrays(gl.POINTS, 0, n);

Points in OpenGL

GL_POINTS

- Draws square pixel region on screen
- One pixel wide by default
- With antialiasing, circular region drawn with smooth edges
- Size controllable with `glPointSize()`
- Easy, efficient way to activate pixels




```
glPointSize(10.0f);
glBegin(GL_POINTS);
  for(int i=0; i<N; i++)
    glVertex3fv(v[i]);
glEnd();
```

[V0.x, V0.y, V1.x, V1.y, V2.x, V2.y, V3.x, V3.y, V4.x, V4.y, V5.x, V5.y, V6.x, V6.y, …]

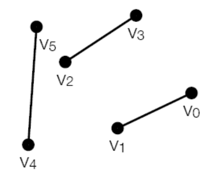
Line 0 Line 1 Line 2

Attribute Vec2 a_Position;

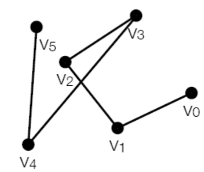
drawArrays(gl.LINES, 0, n/2);

Lines in OpenGL

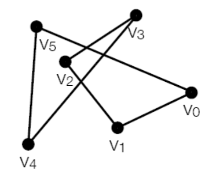
GL_LINES
- Draws lines one pixel wide
- Width can be controlled by `glLineWidth()`
- Successive *pairs* of vertices specify segments



GL_LINE_STRIP
- Like GL_LINES, but successive vertices specify next connected segment in strip



GL_LINE_LOOP
- Like GL_LINE_STRIP, but also connects last and first vertex



[V0.x, V0.y, V1.x, V1.y, V2.x, V2.y, V3.x, V3.y, V4.x, V4.y, V5.x, V5.y, V6.x, V6.y, …]

Triangle 0 Triangle 1

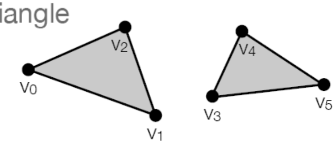
Attribute Vec2 a_Position;

drawArrays(gl.TRIANGLES, 0, n/3);

Triangles in OpenGL

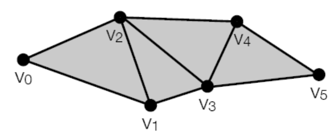
GL_TRIANGLES

- Successive vertex triples specify individual triangles
- Requires three vertices to be emitted for every triangle



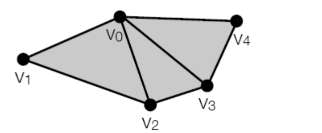
GL_TRIANGLE_STRIP

- First triple specifies first triangle
- Subsequent vertices *each* specify new triangle, along with previous two vertices
- One vertex emitted per triangle in long strips
- But stripifying meshes is nontrivial



GL_TRIANGLE_FAN

- First vertex is center of fan
- Subsequent vertices form ordered bounday
- One vertex emitted per triangle for dense fans
- But few such fans arise in practice



Vertex shader runs for *each* item in the buffer

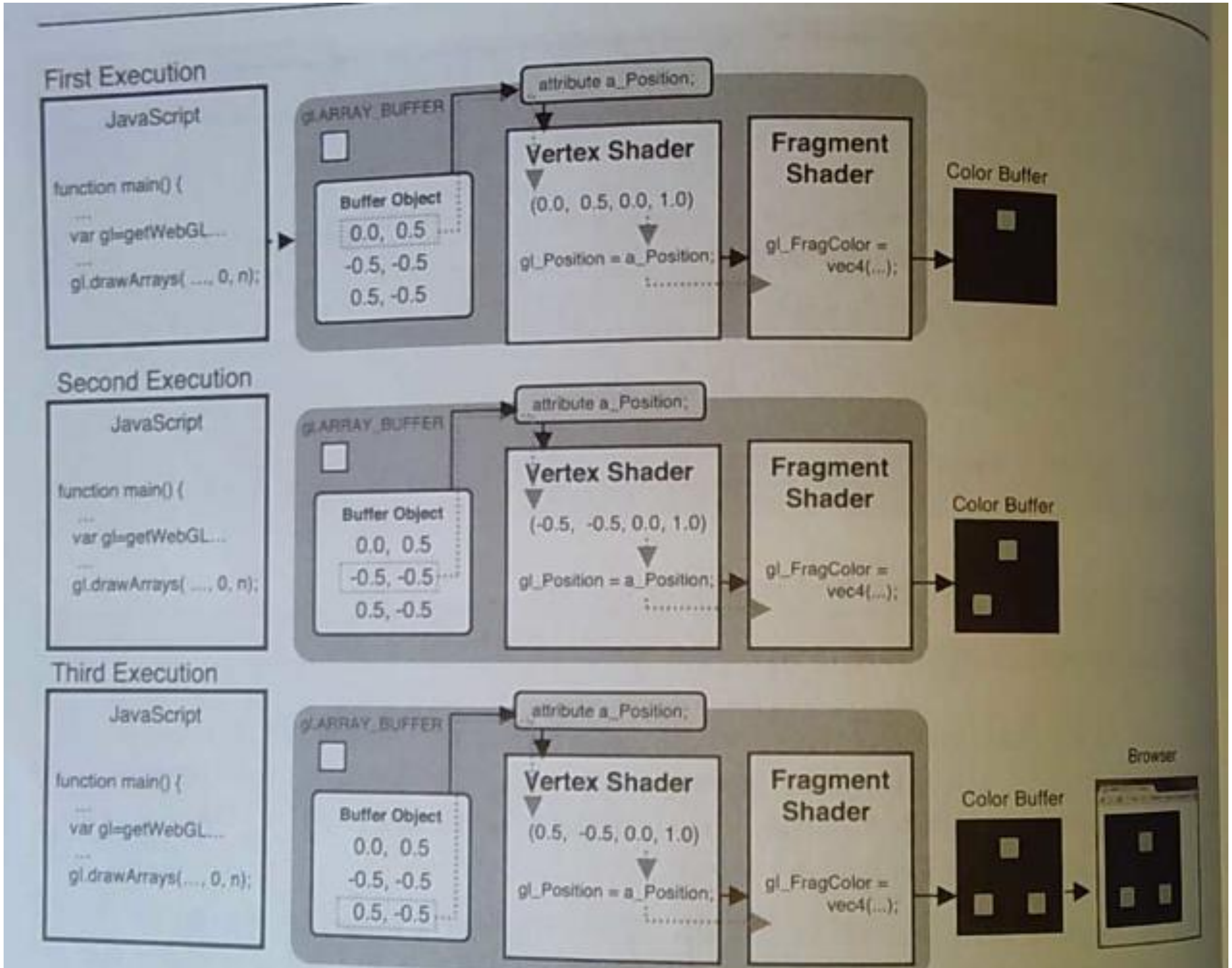


Figure 3.11 How the data in a buffer object is passed to a vertex shader during execution

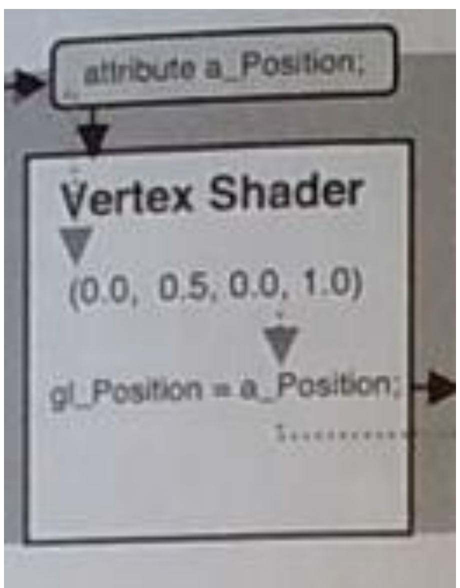
[V0.x, V0.y, V1.x, V1.y, V2.x, V2.y, V3.x, V3.y, V4.x, V4.y, V5.x, V5.y, V6.x, V6.y, …]

Triangle 0

Triangle 1

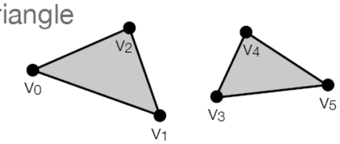
Attribute Vec2 a_Position;

drawArrays(gl.TRIANGLES, 0, n/3);

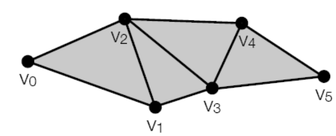


Triangles in OpenGL

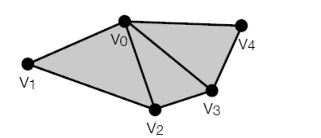
GL_TRIANGLES
- Successive vertex triples specify individual triangles
- Requires three vertices to be emitted for every triangle



GL_TRIANGLE_STRIP
- First triple specifies first triangle
- Subsequent vertices *each* specify new triangle, along with previous two vertices
- One vertex emitted per triangle in long strips
- But stripifying meshes is nontrivial



GL_TRIANGLE_FAN
- First vertex is center of fan
- Subsequent vertices form ordered bounday
- One vertex emitted per triangle for dense fans
- But few such fans arise in practice



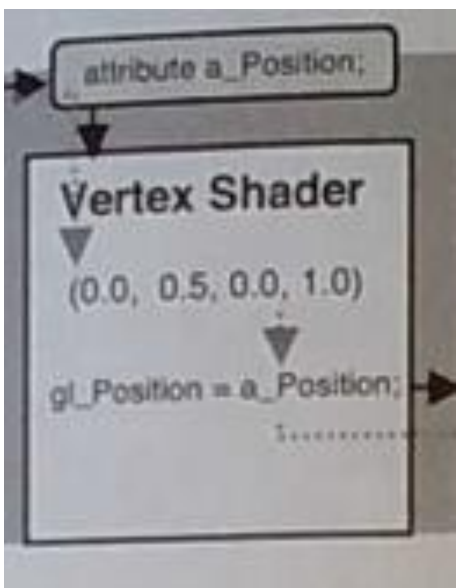
[V0.x, V0.y, V1.x, V1.y, V2.x, V2.y, V3.x, V3.y, V4.x, V4.y, V5.x, V5.y, V6.x, V6.y, …]

Triangle 0

Triangle 1

Attribute Vec2 a_Position;

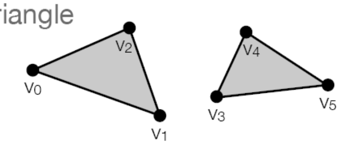
drawArrays(gl.TRIANGLES, 0, n/3);



Triangles in OpenGL

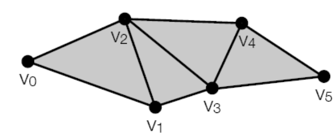
GL_TRIANGLES

- Successive vertex triples specify individual triangles
- Requires three vertices to be emitted for every triangle



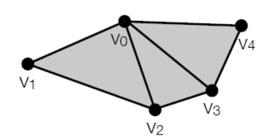
GL_TRIANGLE_STRIP

- First triple specifies first triangle
- Subsequent vertices *each* specify new triangle, along with previous two vertices
- One vertex emitted per triangle in long strips
- But stripifying meshes is nontrivial



GL_TRIANGLE_FAN

- First vertex is center of fan
- Subsequent vertices form ordered bounday
- One vertex emitted per triangle for dense fans
- But few such fans arise in practice



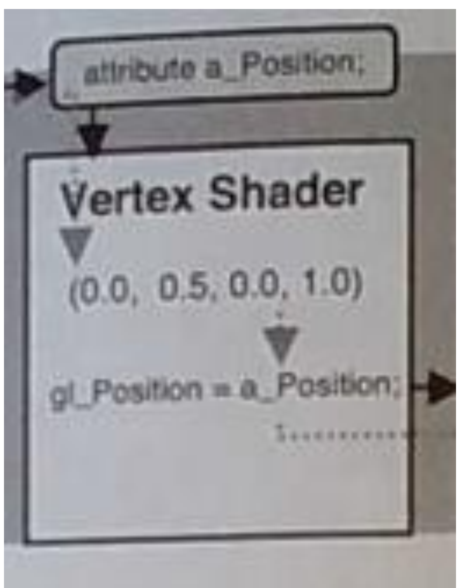
[V0.x, V0.y, V1.x, V1.y, V2.x, V2.y, V3.x, V3.y, V4.x, V4.y, V5.x, V5.y, V6.x, V6.y, …]

Triangle 0

Triangle 1

Attribute Vec2 a_Position;

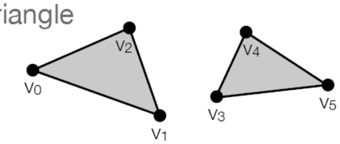
drawArrays(gl.TRIANGLES, 0, n/3);



Triangles in OpenGL

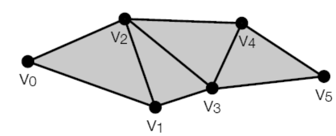
GL_TRIANGLES

- Successive vertex triples specify individual triangles
- Requires three vertices to be emitted for every triangle



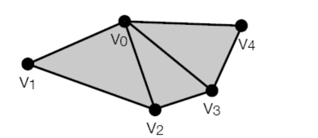
GL_TRIANGLE_STRIP

- First triple specifies first triangle
- Subsequent vertices *each* specify new triangle, along with previous two vertices
- One vertex emitted per triangle in long strips
- But stripifying meshes is nontrivial



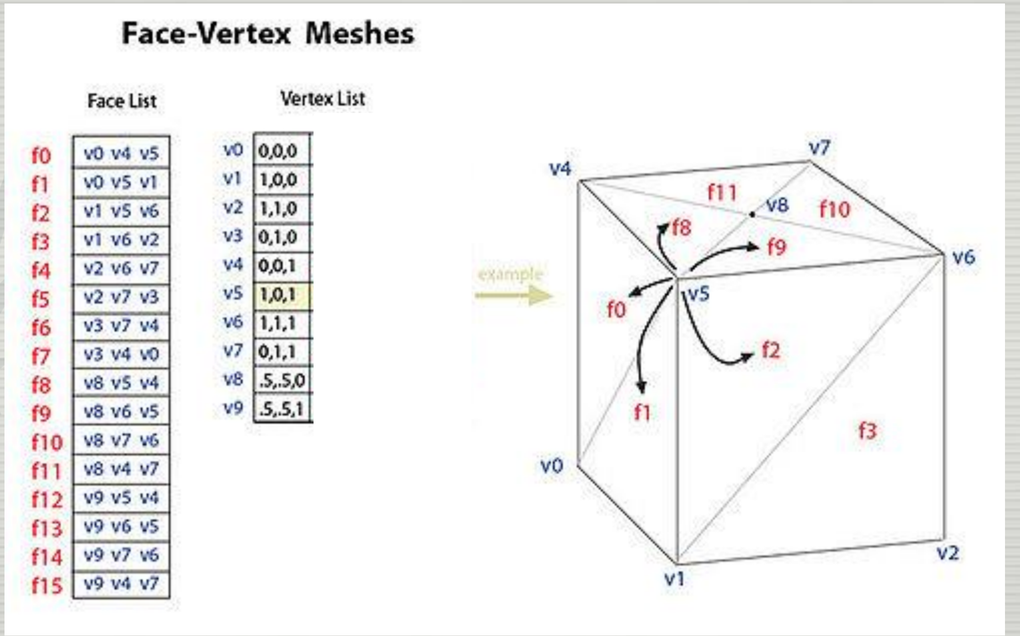
GL_TRIANGLE_FAN

- First vertex is center of fan
- Subsequent vertices form ordered bounday
- One vertex emitted per triangle for dense fans
- But few such fans arise in practice



Face-Vertex Meshes (FV)

- List of faces defined by vertex indices



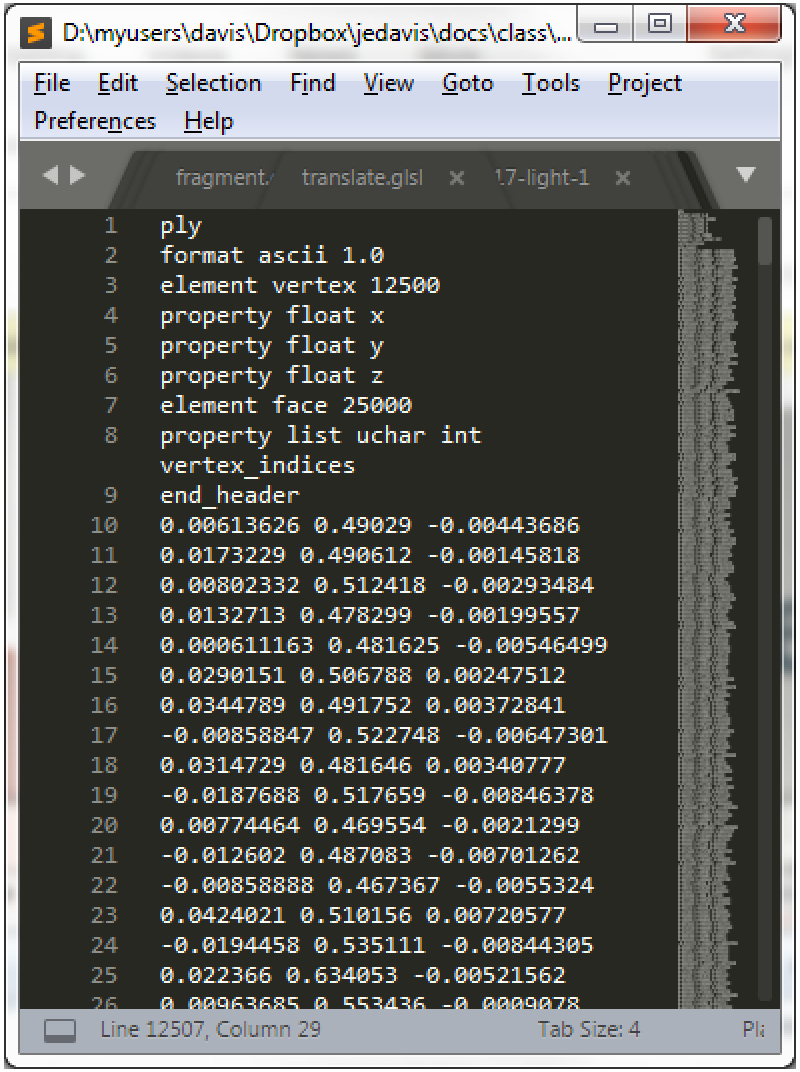
(image courtesy of Wikipedia)

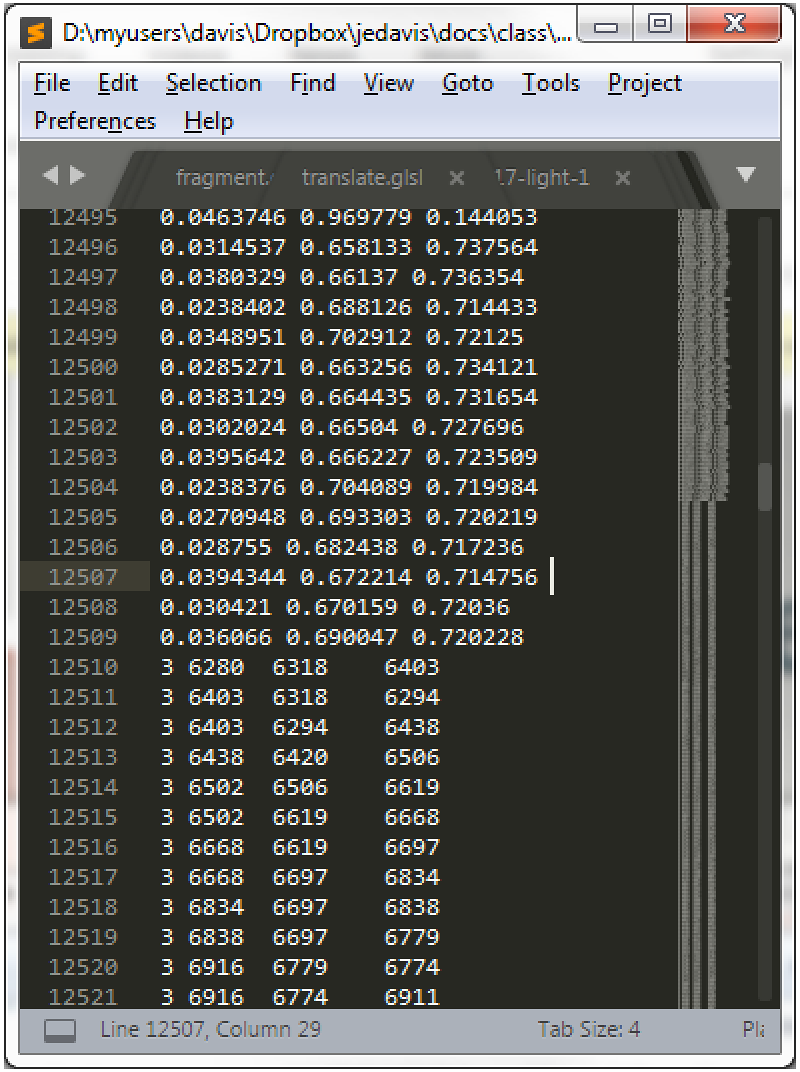
~~drawArrays(gl.TRIANGLES, 0, n/3);~~
drawElements(…);

3D Scene/Model File Formats

- Wavefront OBJ (.obj)
- 3DS Max (.3ds)
- Geomview OFF (Object File Format) (.off)
- PLY (.ply) for scanned data
- … and more

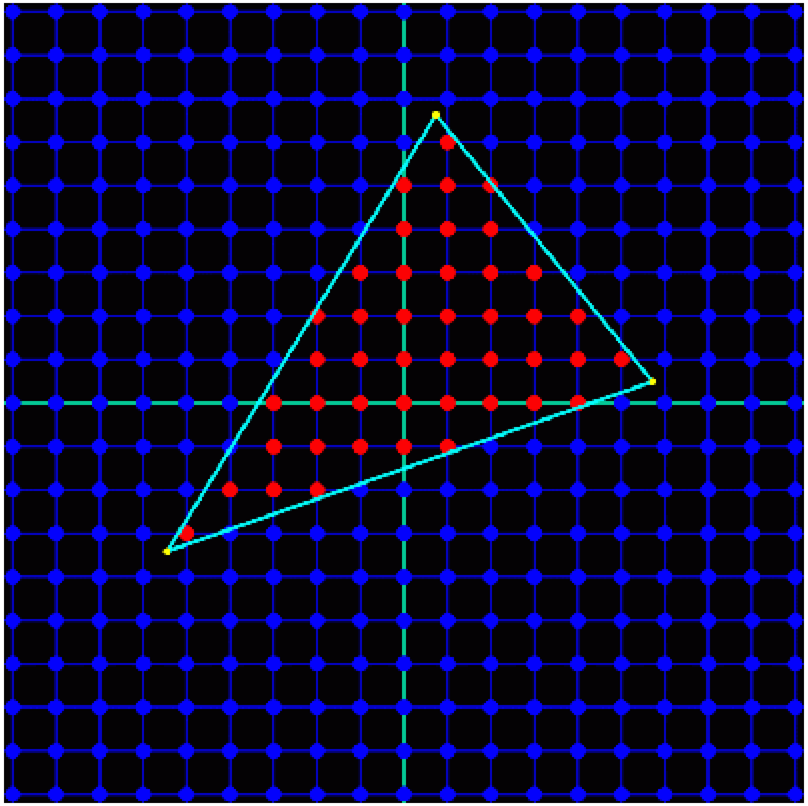
Example data in a 3D file (.ply)





Rasterization

Rasterization = Turn on all pixels inside the triangle

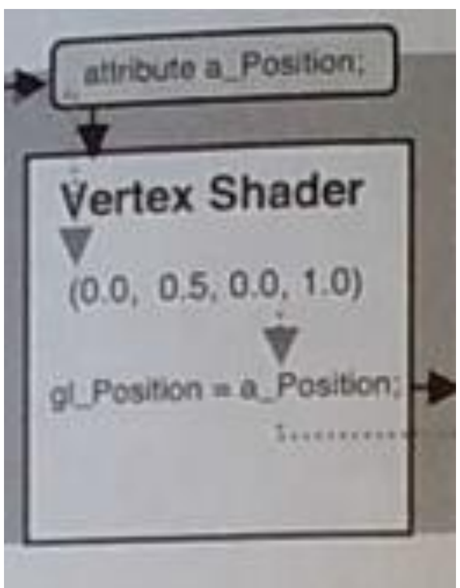


[V0.x, V0.y, V1.x, V1.y, V2.x, V2.y, V3.x, V3.y, V4.x, V4.y, V5.x, V5.y, V6.x, V6.y, …]

Triangle 0 Triangle 1

Attribute Vec2 a_Position;

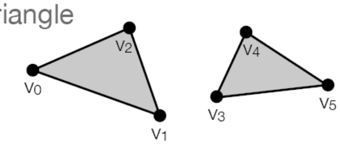
drawArrays(gl.TRIANGLES, 0, n/3);



Triangles in OpenGL

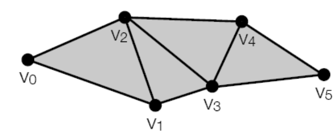
GL_TRIANGLES

- Successive vertex triples specify individual triangles
- Requires three vertices to be emitted for every triangle



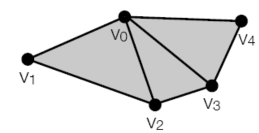
GL_TRIANGLE_STRIP

- First triple specifies first triangle
- Subsequent vertices *each* specify new triangle, along with previous two vertices
- One vertex emitted per triangle in long strips
- But stripifying meshes is nontrivial



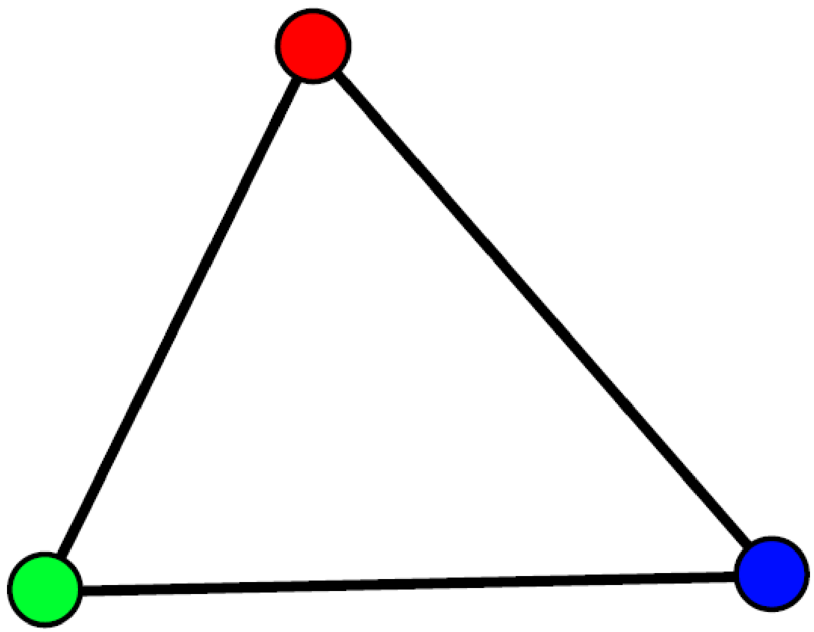
GL_TRIANGLE_FAN

- First vertex is center of fan
- Subsequent vertices form ordered bounday
- One vertex emitted per triangle for dense fans
- But few such fans arise in practice



How is shading done in OpenGL?

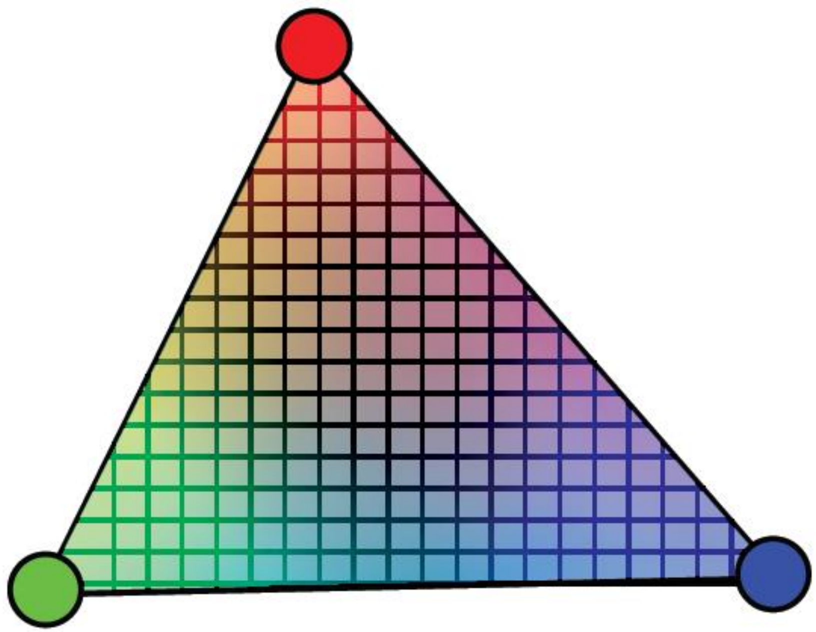
1. Attributes are specified on vertices.



How is shading done in OpenGL?

2. Attributes are interpolated across triangles by the rasterizer

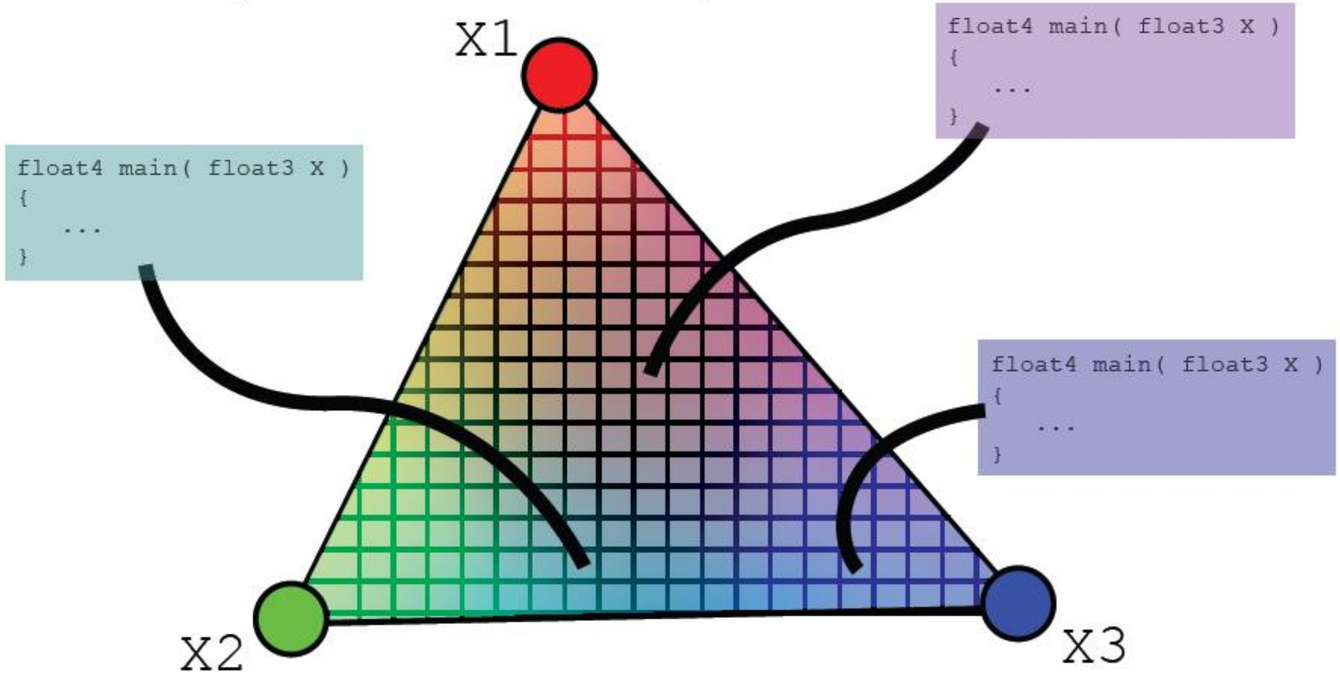
(see appendix for details)



Rasterizer also breaks the triangle into “fragments.”

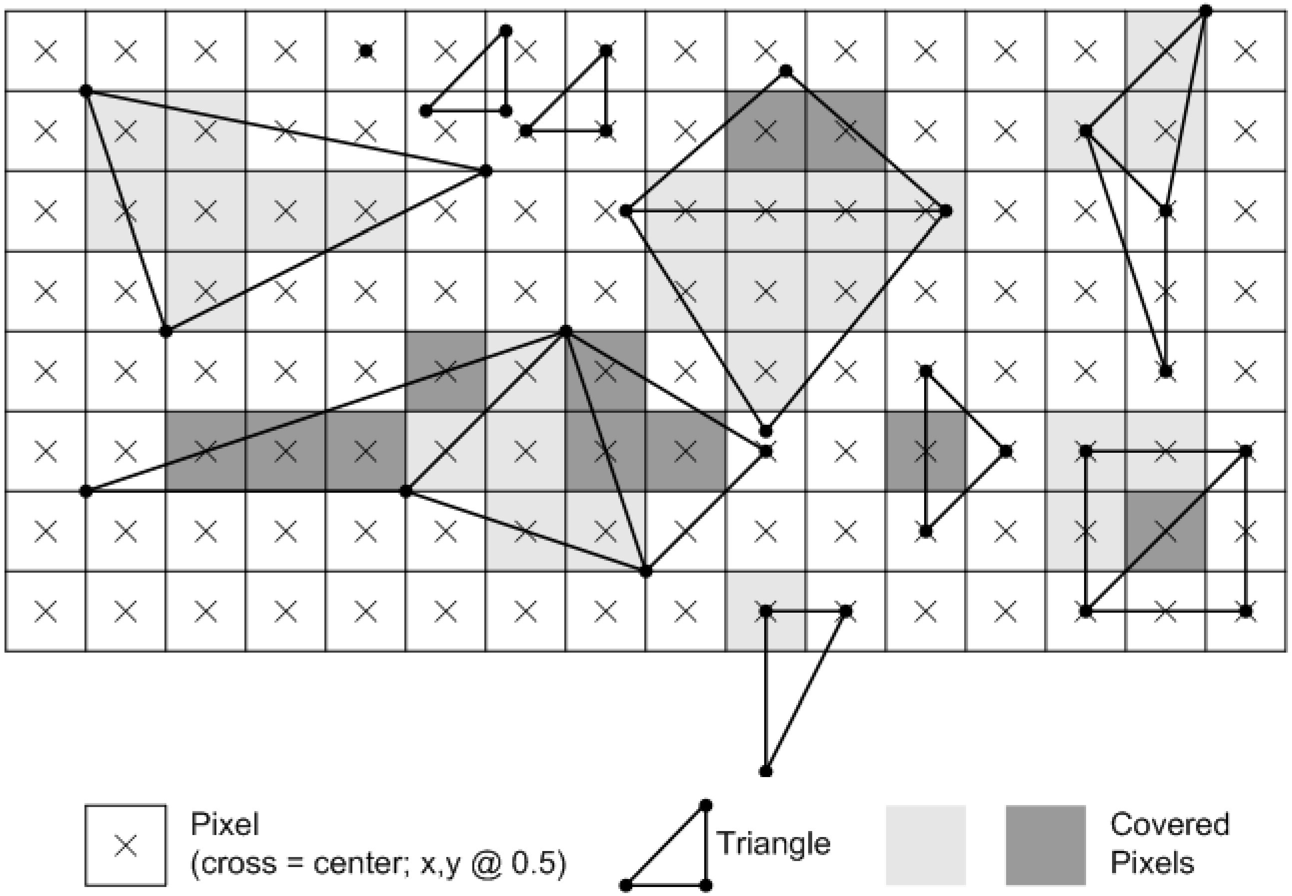
How is shading done in OpenGL?

3. Each fragment runs the shader using interpolated values as inputs.



Exact same routine, different inputs.

Top-Left Rasterization Rule

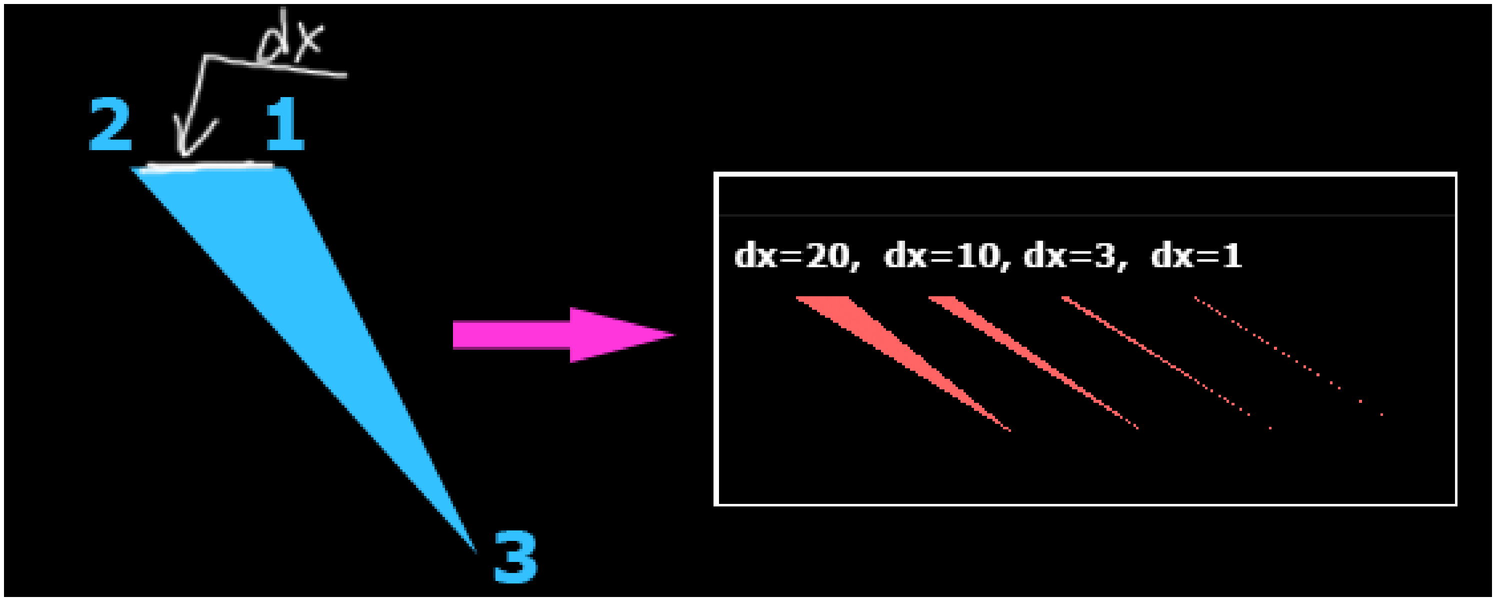


[Rasterization Rules (Direct3D 10) – MSDN]

2

I'm using TriangleList to output my primitives. Most all of the time I need to draw rectangles, triangles, circles. From time to time I need to draw very thin triangles (width=2px for example). I thought it should look like a line (almost a line) but it looks like separate points :)

Following picture shows what I'm talking about:



First picture at the left side shows how do I draw a rectangle (counter clockwise, from top right corner). And then you can see the "width" of the rectangle which I call "dx".

How to avoid this behavior? I would it looks like a straight (almost straight) line, not as points :)

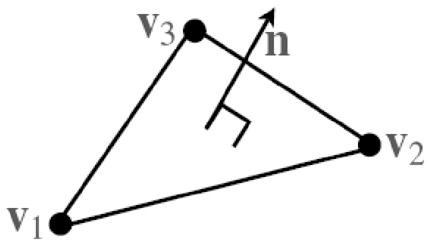
opengl gl-triangle-strip

Normals

Triangle Normals

Per-Triangle

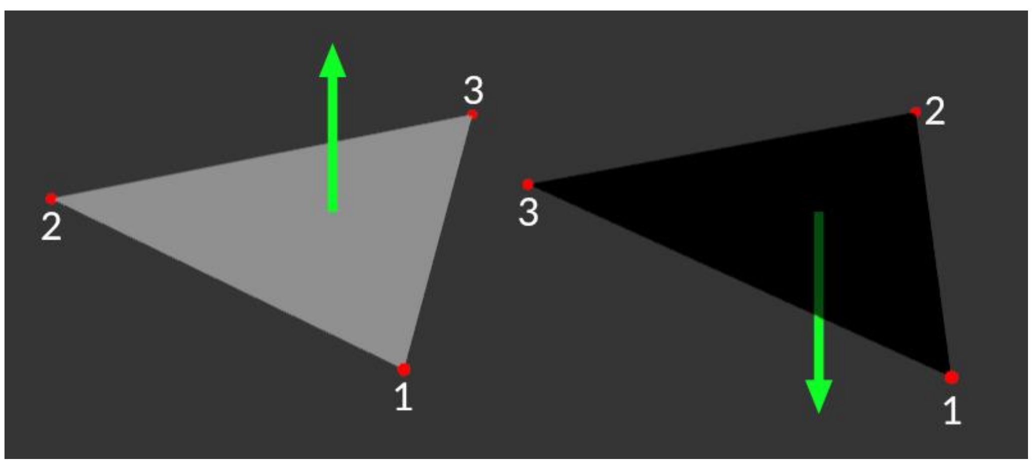
- Triangle defines unique plane:
- Can easily compute **unit** normal vector from vertices:



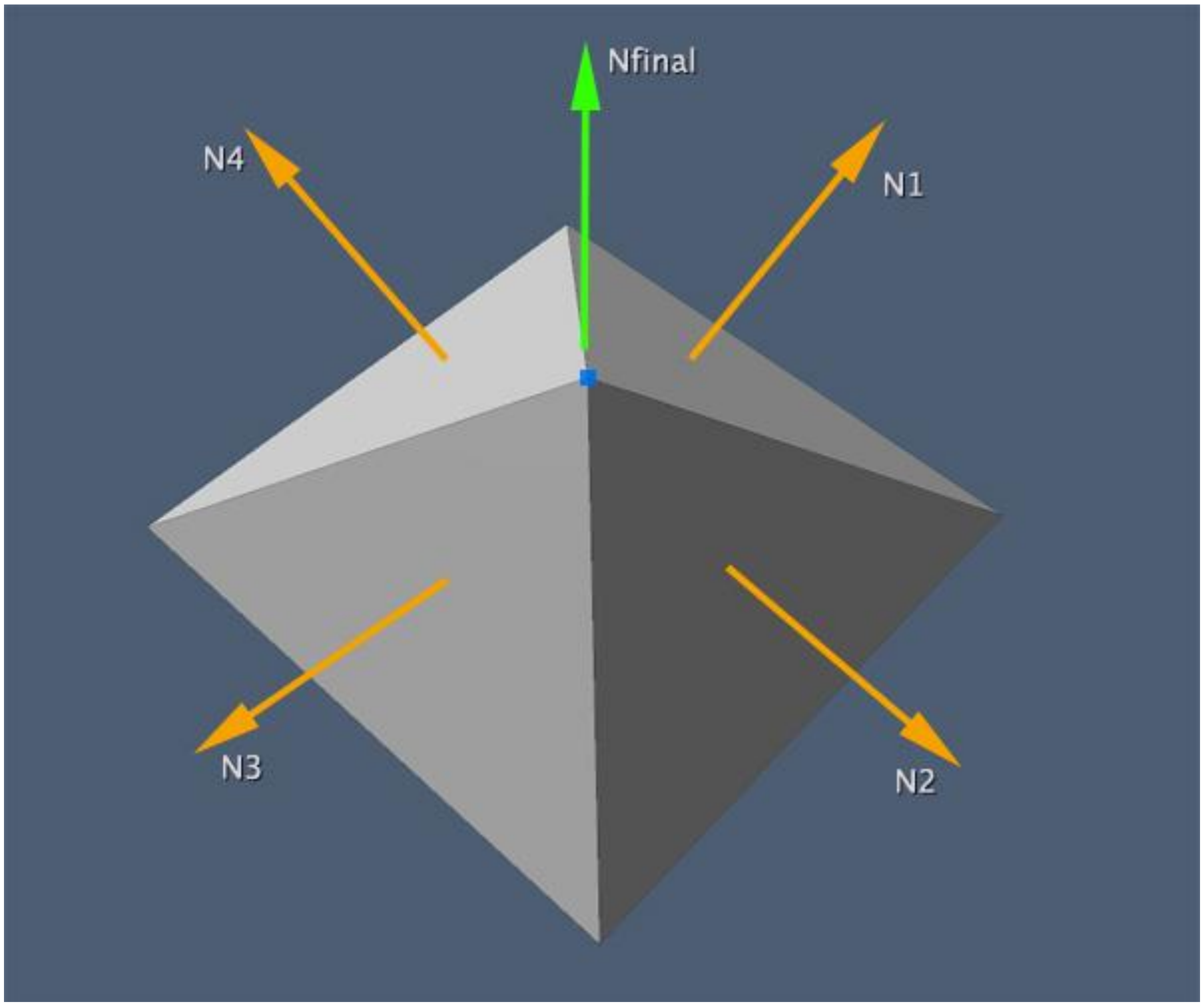
$$\mathbf{a} = \mathbf{v}_2 - \mathbf{v}_1$$
$$\mathbf{b} = \mathbf{v}_3 - \mathbf{v}_1$$

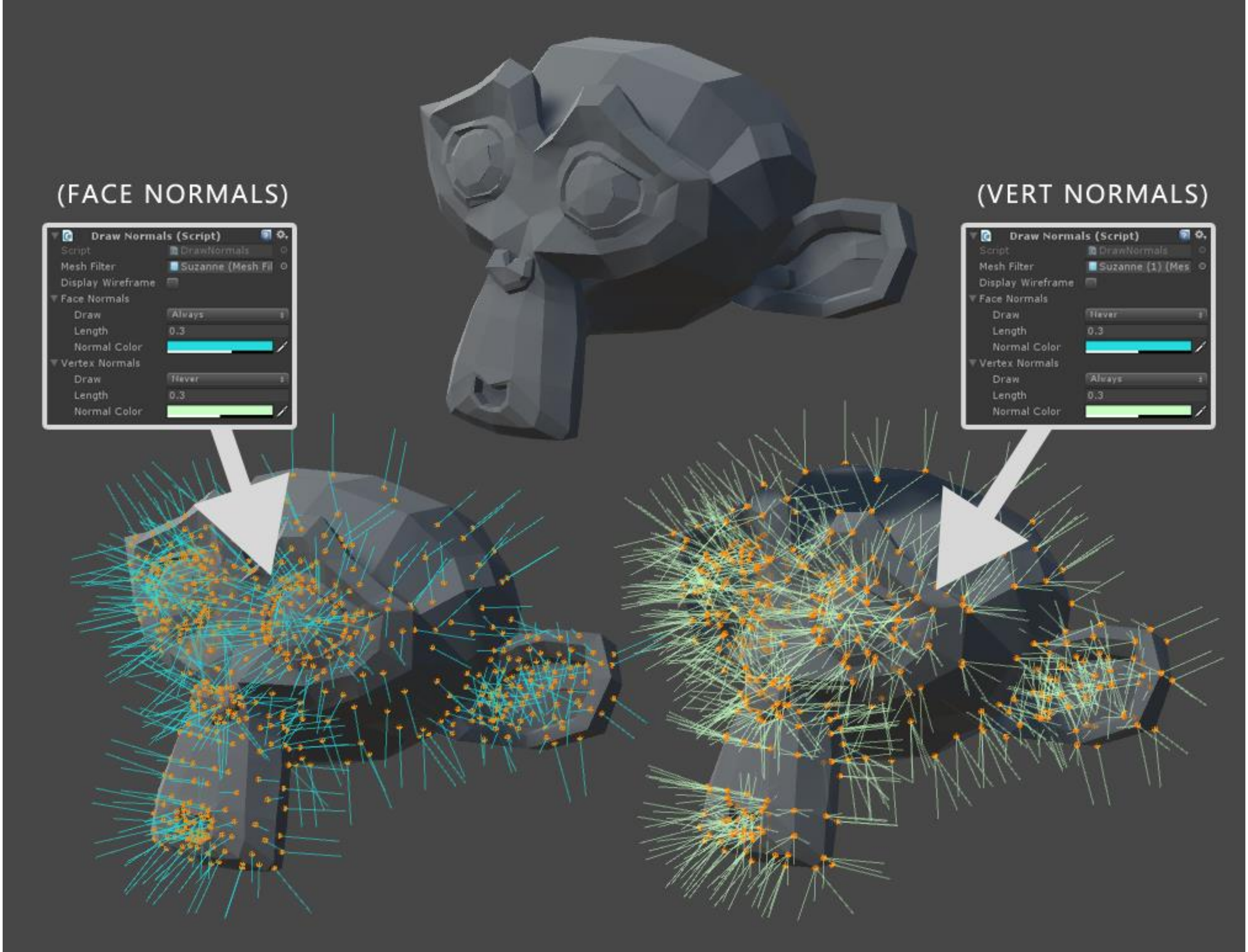
$$\mathbf{n} = \frac{\mathbf{a} \times \mathbf{b}}{\|\mathbf{a} \times \mathbf{b}\|}$$

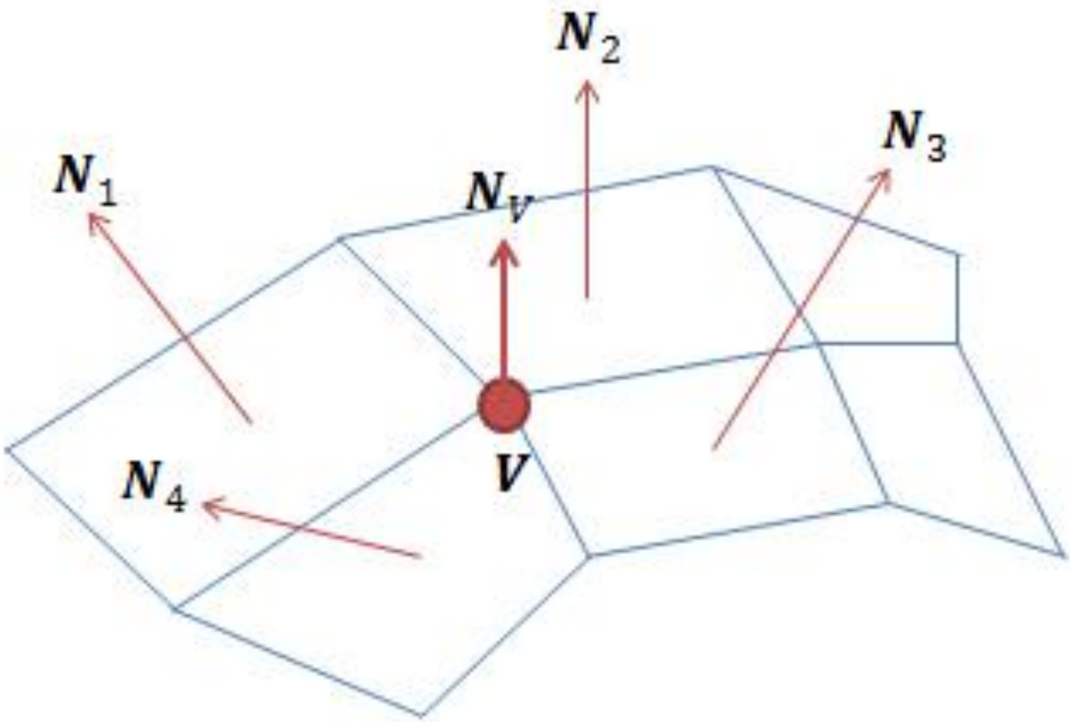
- Orientation depends on vertex order (clockwise yields $-\mathbf{n}$)



How can a vertex have a normal?

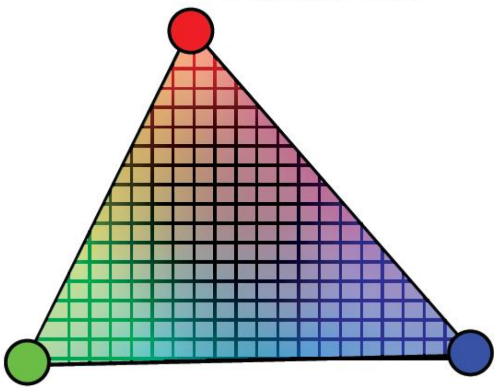



(FACE NORMALS)
(VERT NORMALS)
Draw Normals (Script)
Script DrawNormals
Mesh Filter Suzanne (Mesh Fil
Display Wireframe
Face Normals
Draw Always
Length 0.3
Normal Color
Vertex Normals
Draw Never
Length 0.3
Normal Color
Draw Normals (Script)
Script DrawNormals
Mesh Filter Suzanne (1) (Mes
Display Wireframe
Face Normals
Draw Never
Length 0.3
Normal Color
Vertex Normals
Draw Always
Length 0.3
Normal Color

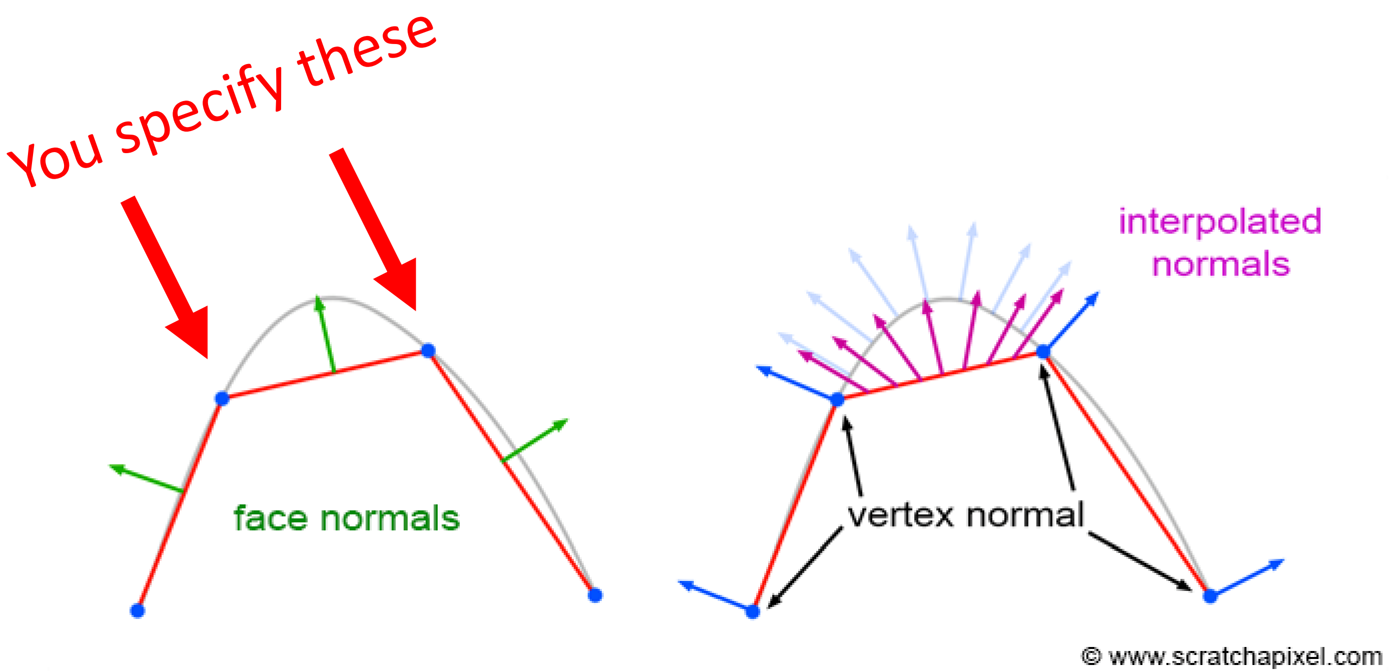


$$N_V = \frac{\sum_{k=1}^{n} N_k}{\left|\sum_{k=1}^{n} N_k\right|}$$

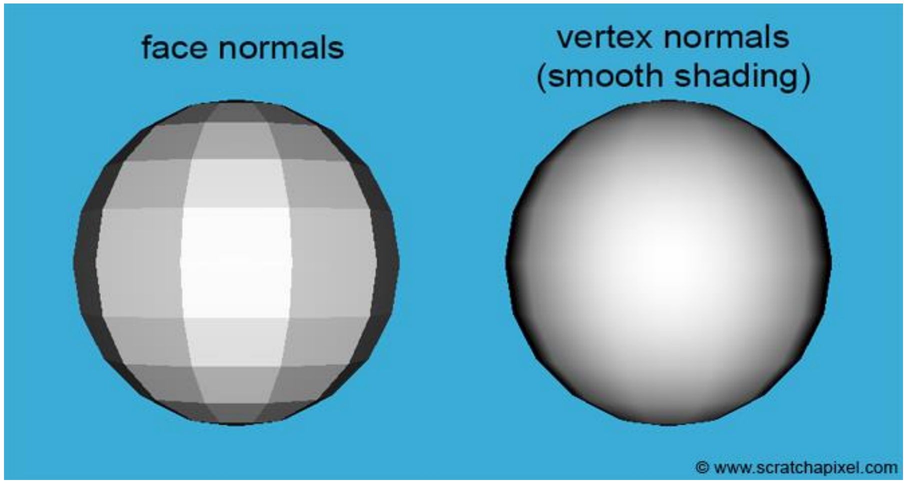
Interpolating color

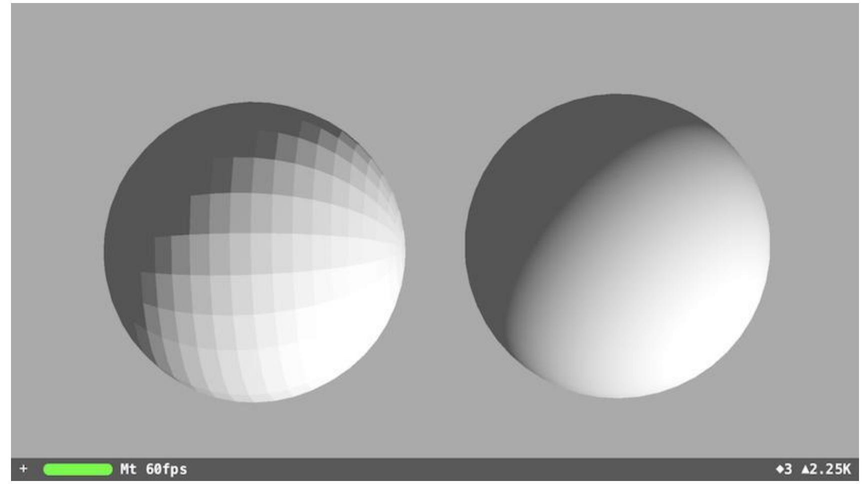


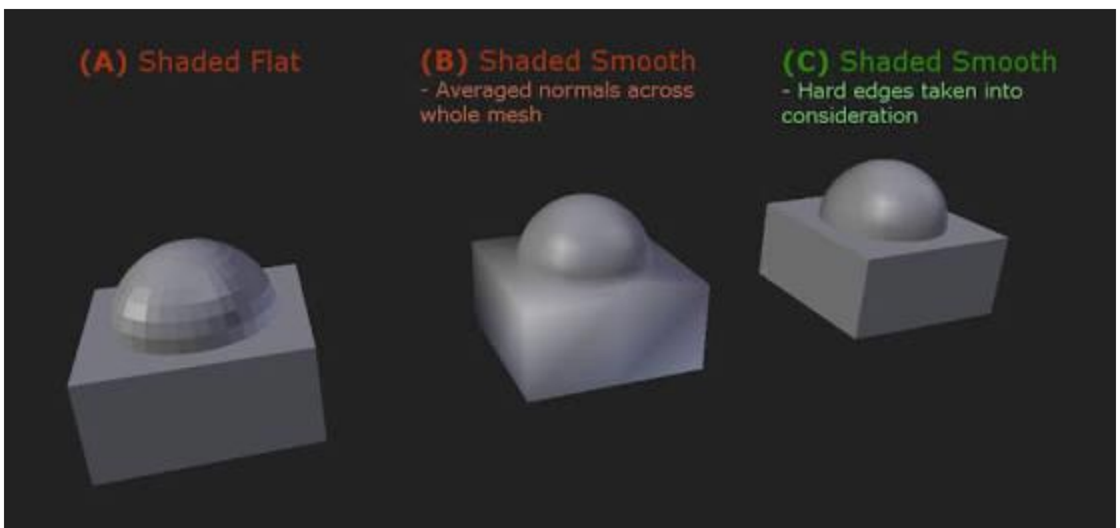
Interpolating normals



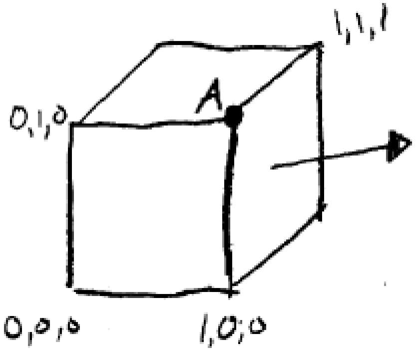
Per face vs per vertex normals







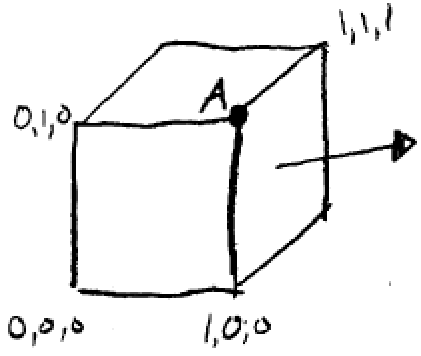
Q about Normals



What is the per-polygon normal shown?

(A) 1,1,1
(B) 0,0,1
(C) 1,0,0
(D) 0,1,0
(E) Don’t know

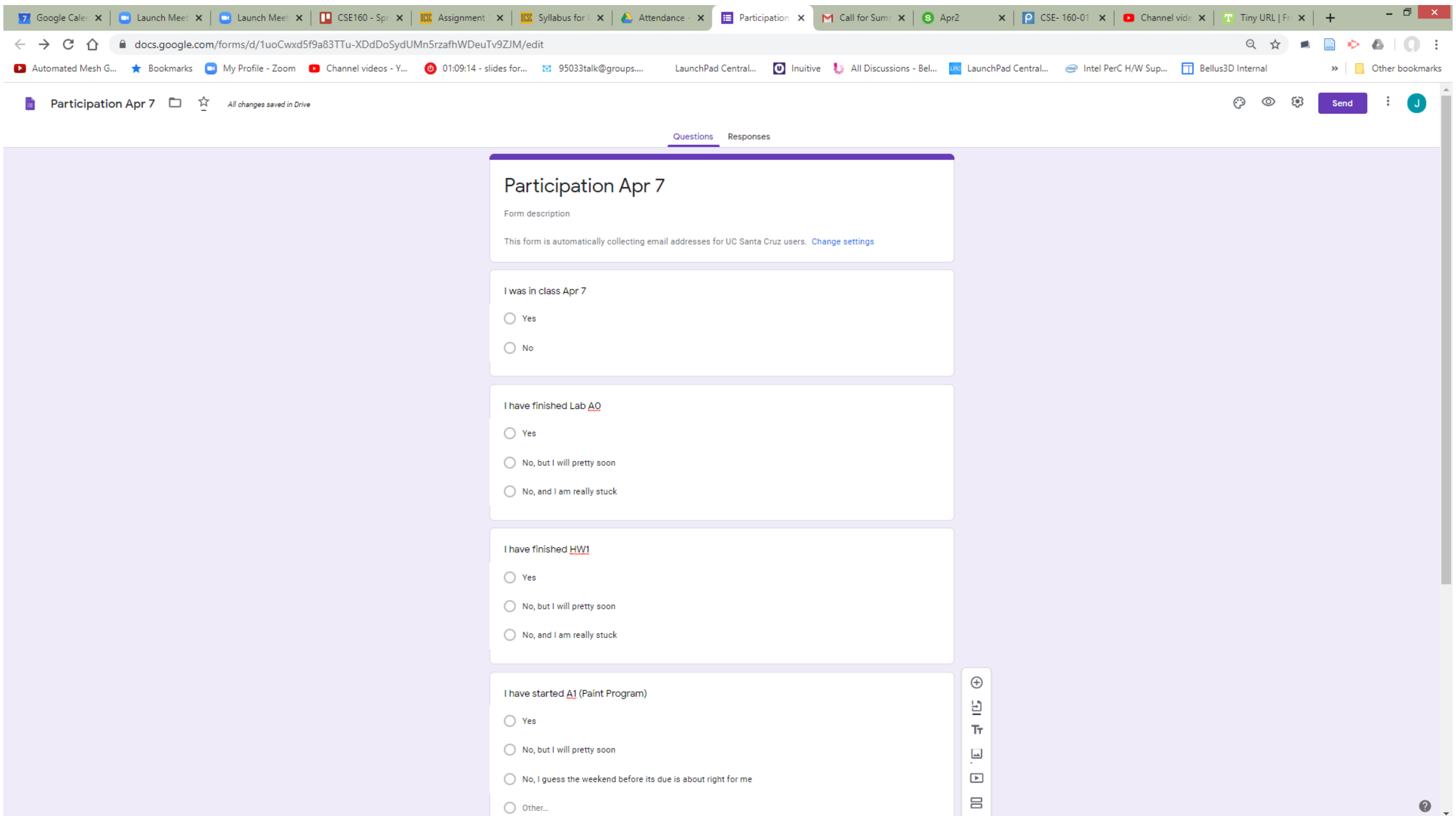
Q about Normals



What is the per-vertex normal at point A?

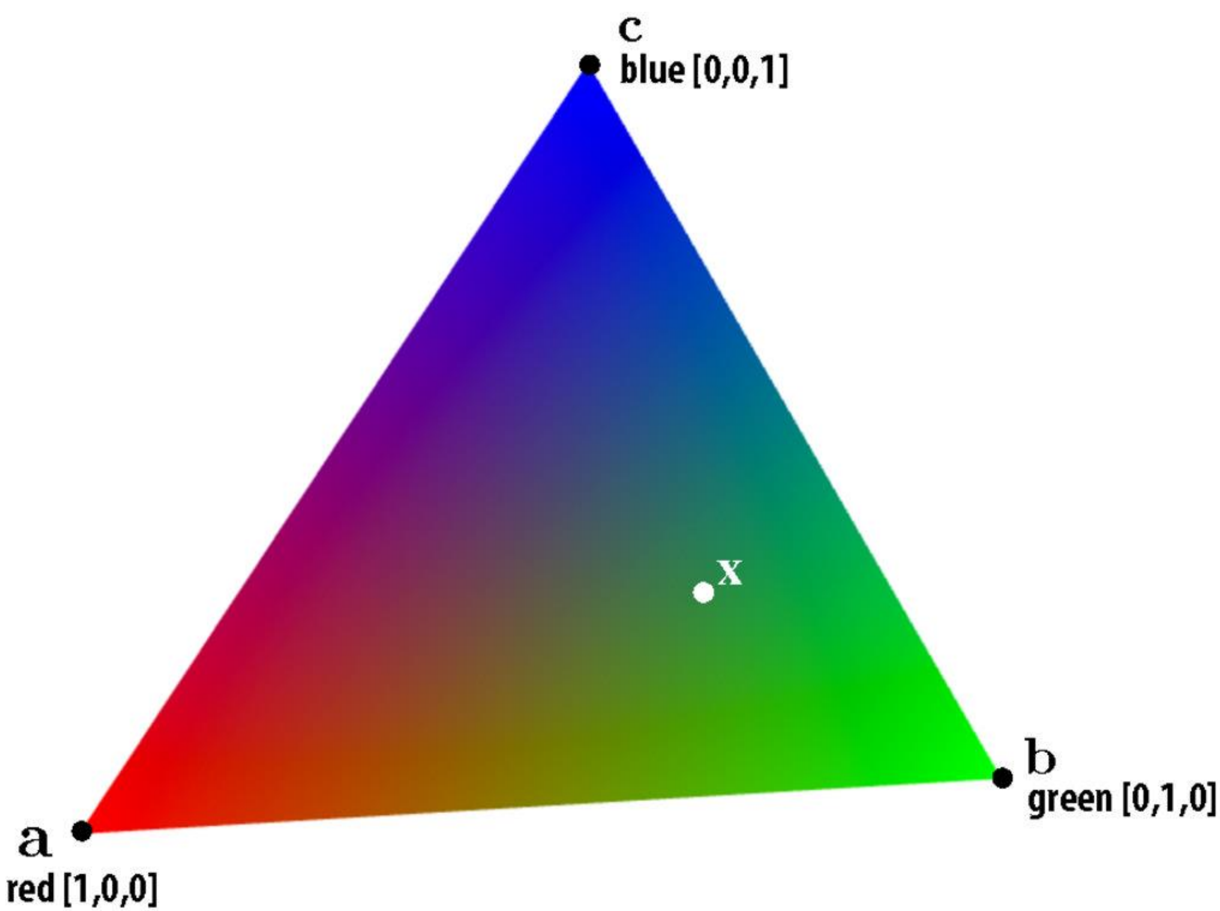
(A) 1,1,1
(B) 1,1,-1
(C) 1/sqrt(3), 1/sqrt(3), 1/sqrt(3)
(D) -1/sqrt(3), 1/sqrt(3),1/sqrt(3)
(E) Don’t know

http://tiny.cc/160108



Interpolation

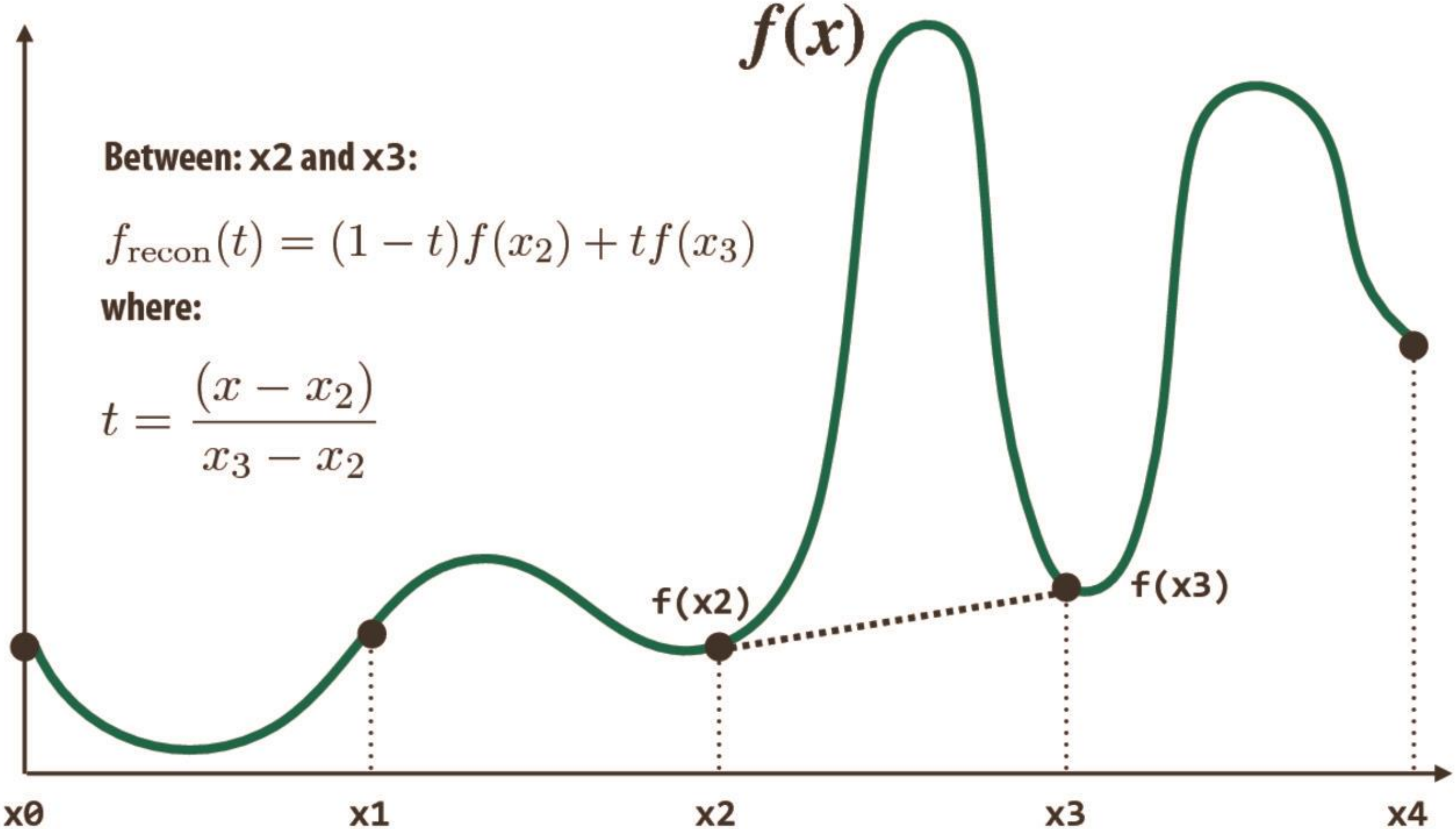
Consider sampling color(x,y)



What is the triangle's color at the point $\mathbf{x}$?

Review: interpolation in 1D

$f_{recon}(x)$ = linear interpolation between values of two closest samples to x



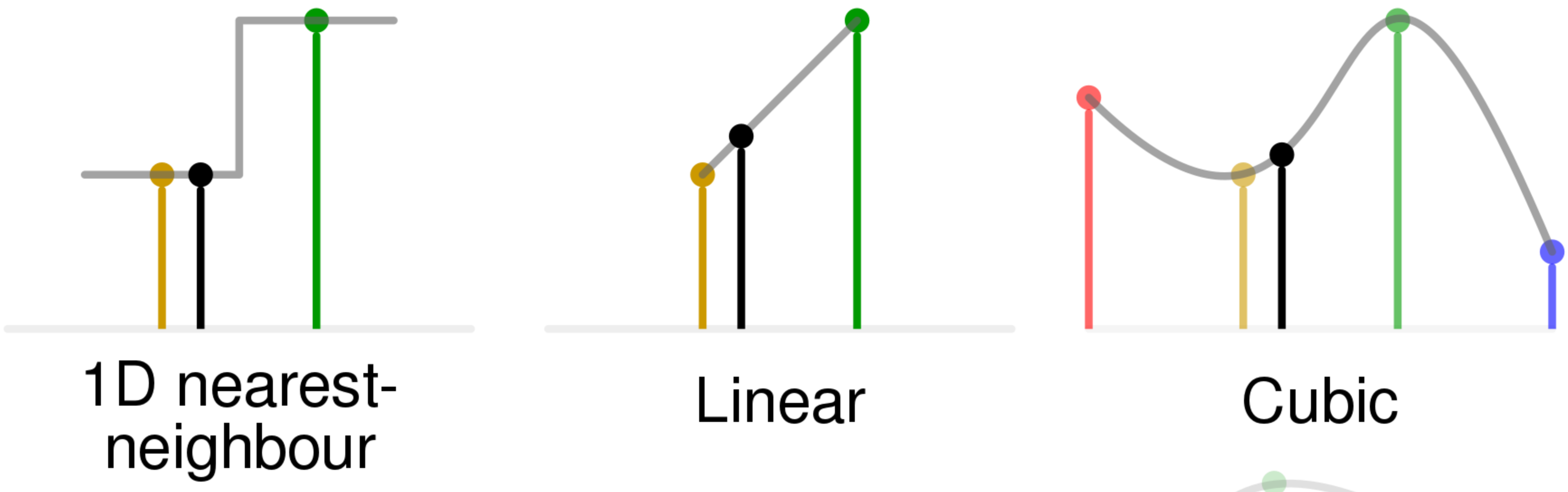
Between: x2 and x3:

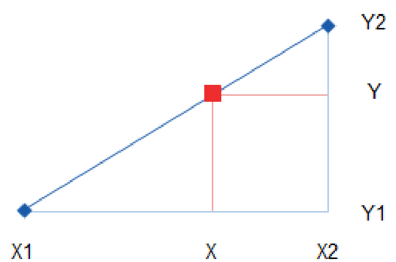
$$f_{\text{recon}}(t) = (1-t)f(x_2) + tf(x_3)$$

where:

$$t = \frac{(x - x_2)}{x_3 - x_2}$$

Linear interpolation

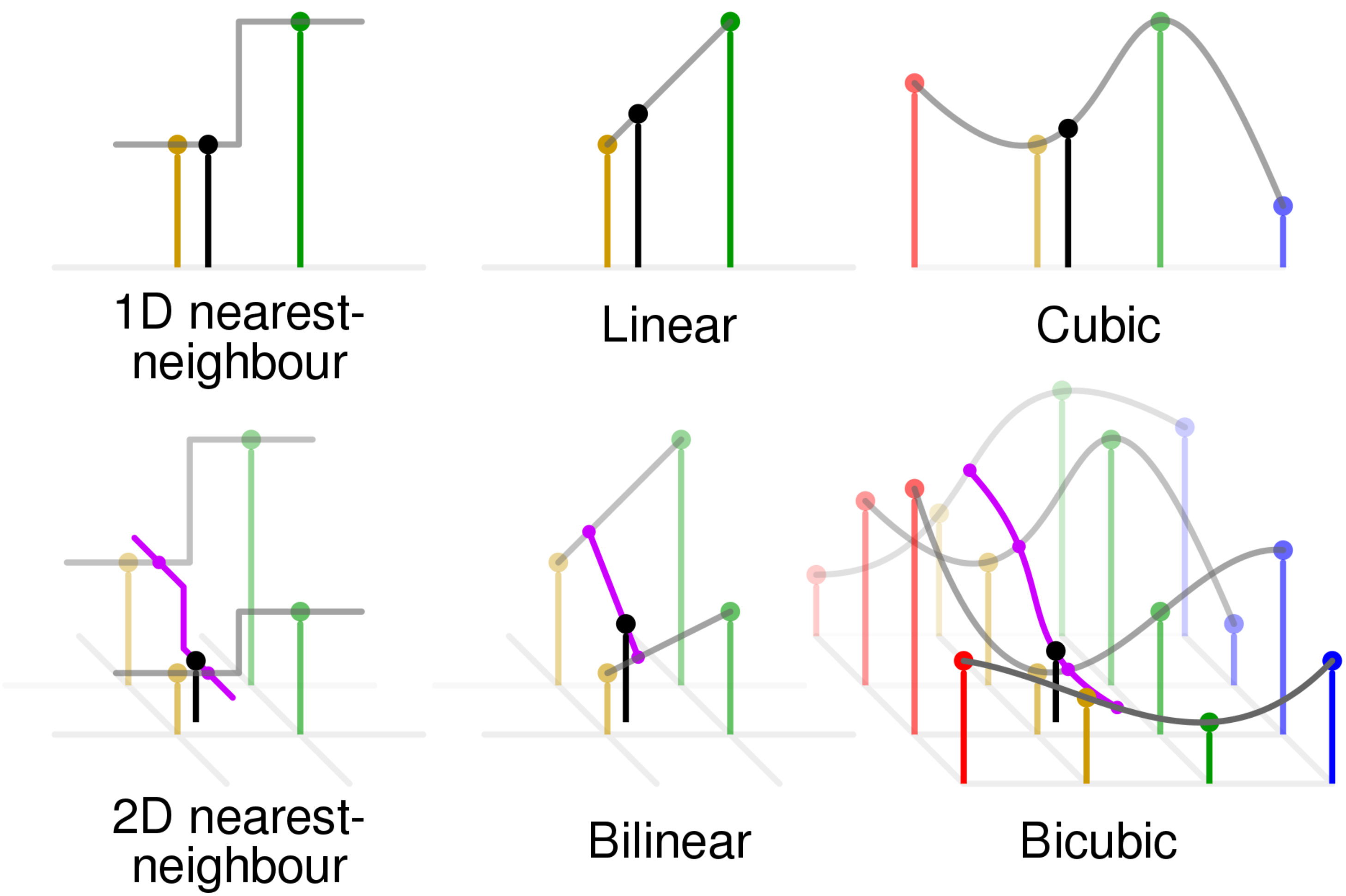




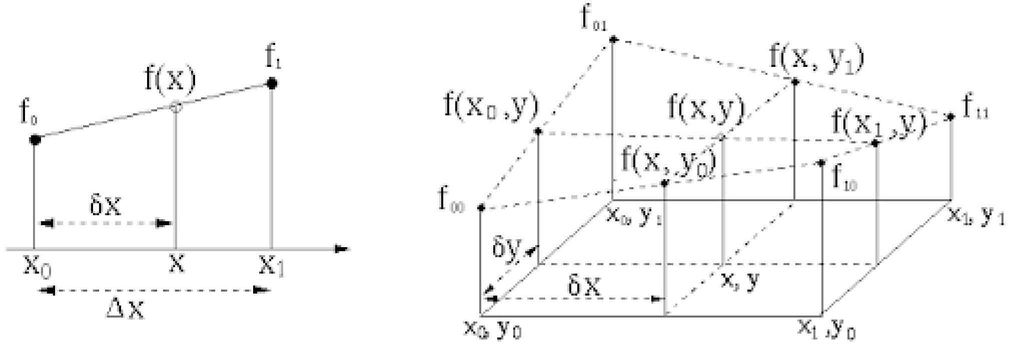
$$\frac{(X - X1)}{(X2 - X1)} = \frac{(Y - Y1)}{(Y2 - Y1)}$$

$$Y = Y1 + (X - X1)\frac{(Y2 - Y1)}{(X2 - X1)}$$

Bi-linear interpolation

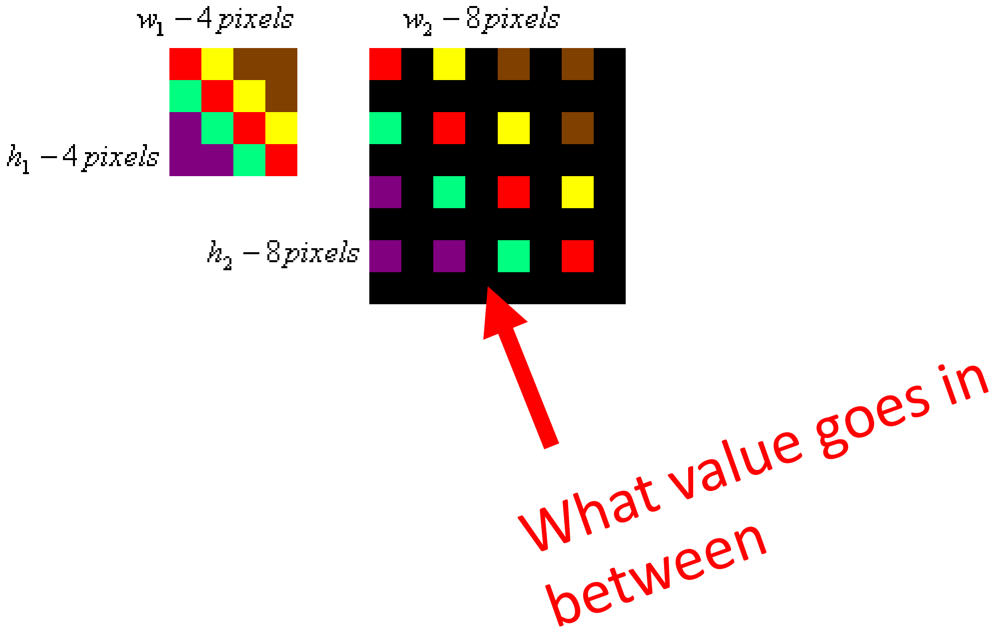


Bi-linear interpolation



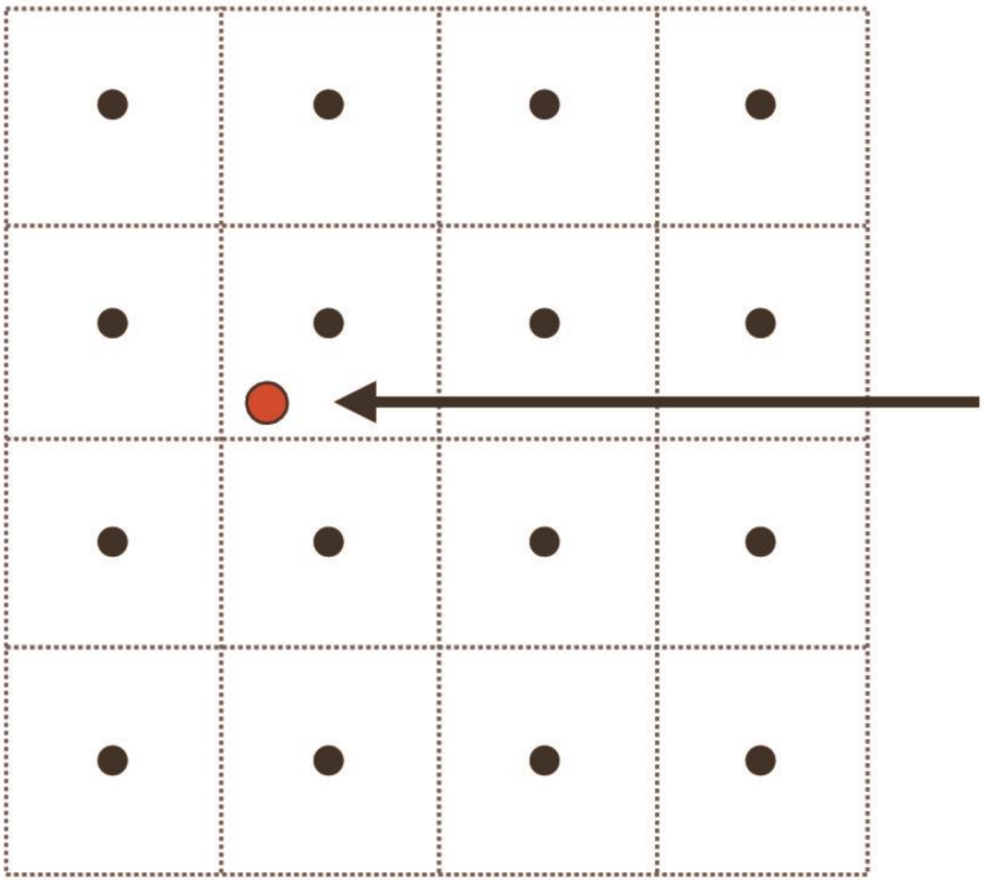
Interpolation is not just about surfaces, used heavily in images

Suppose you start with the smallest image and need a big one?


$w_1 - 4\,pixels$
$h_1 - 4\,pixels$
$w_2 - 8\,pixels$
$h_2 - 8\,pixels$
What value goes in between

Nearest, bilinear, b-spline interpolation

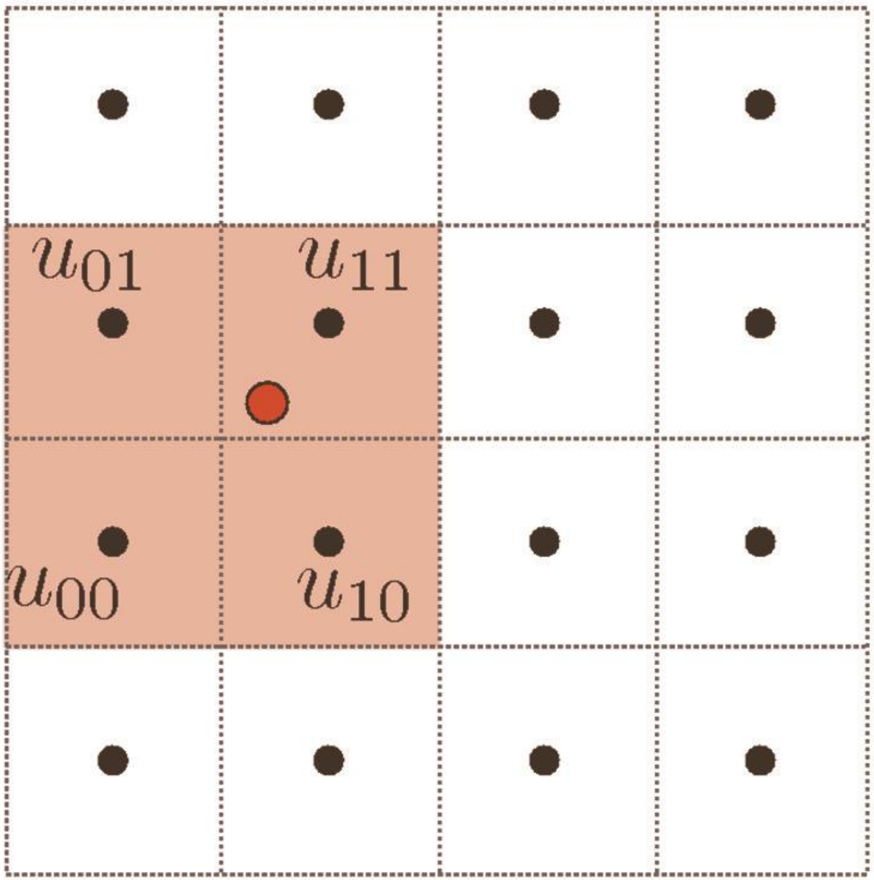
Bilinear Filtering



Want to sample texture value f(x,y) at red point

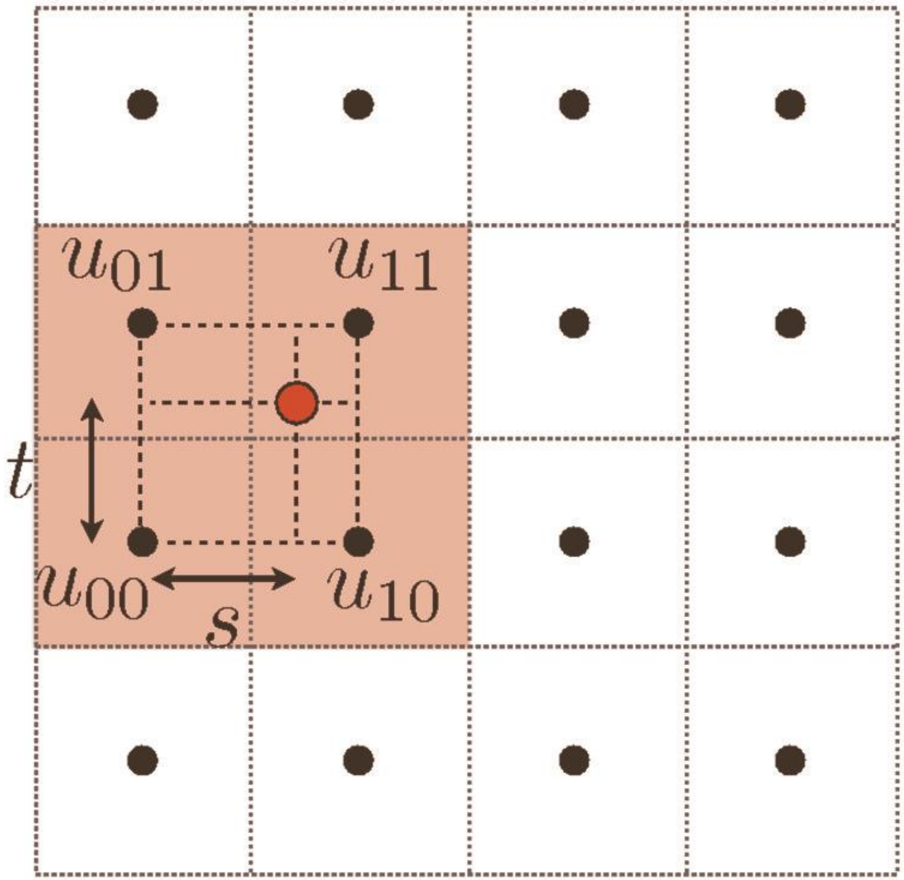
Black points indicate texture sample locations

Bilinear filtering



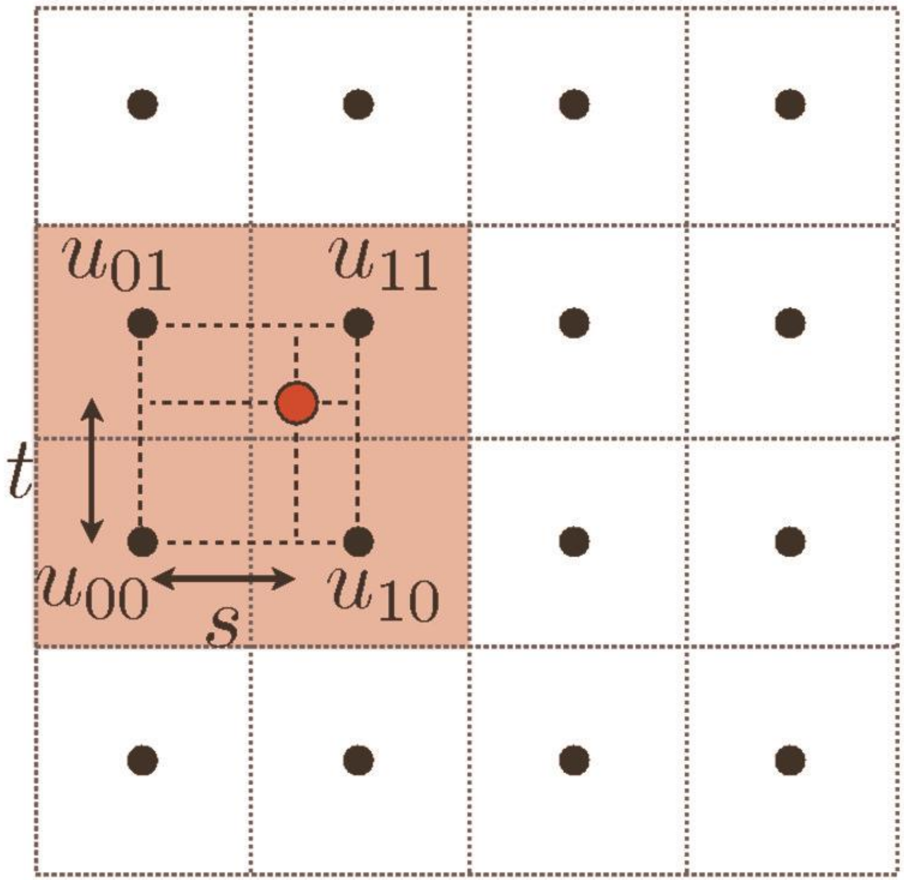
Take 4 nearest sample locations, with texture values as labeled.

Bilinear filtering



And fractional offsets, (s,t) as shown

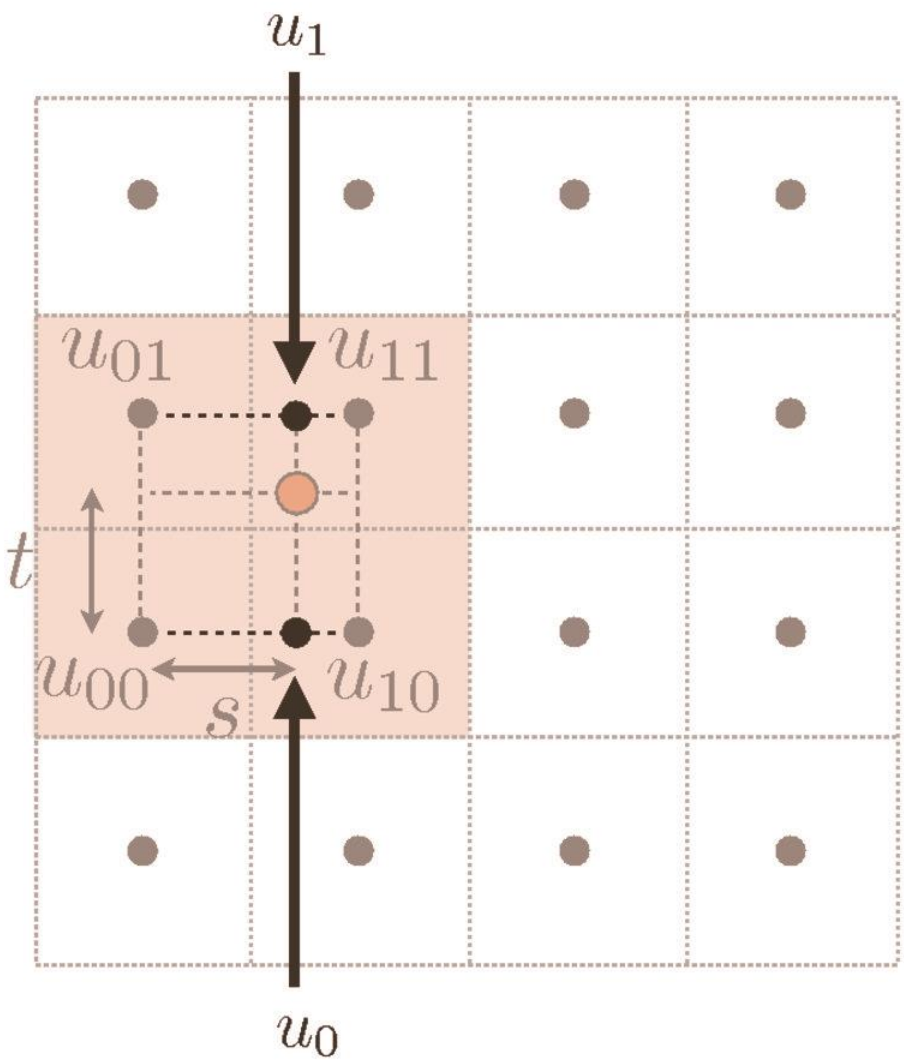
Bilinear filtering



Linear interpolation (1D)

$$\mathrm{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Bilinear filtering



Linear interpolation (1D)

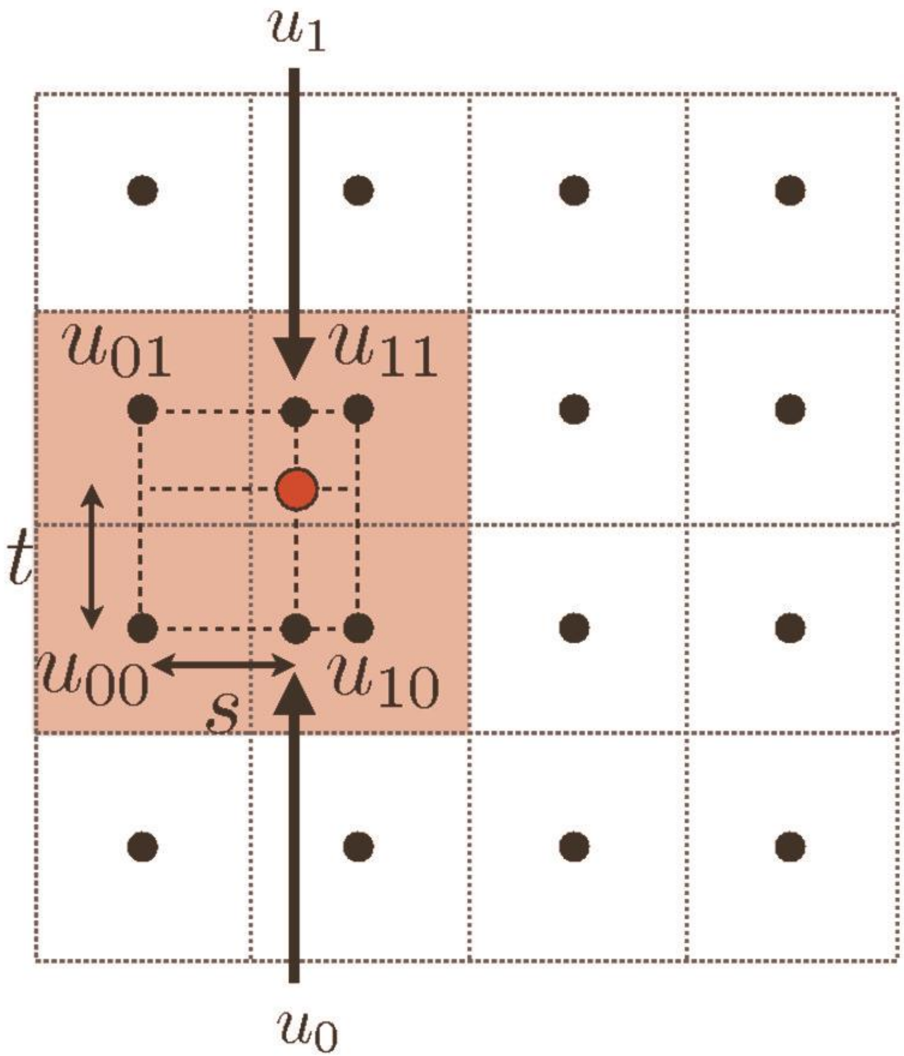
$$\mathrm{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Two helper lerps (horizontal)

$$u_0 = \mathrm{lerp}(s, u_{00}, u_{10})$$

$$u_1 = \mathrm{lerp}(s, u_{01}, u_{11})$$

Bilinear filtering



Linear interpolation (1D)

$$\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Two helper lerps

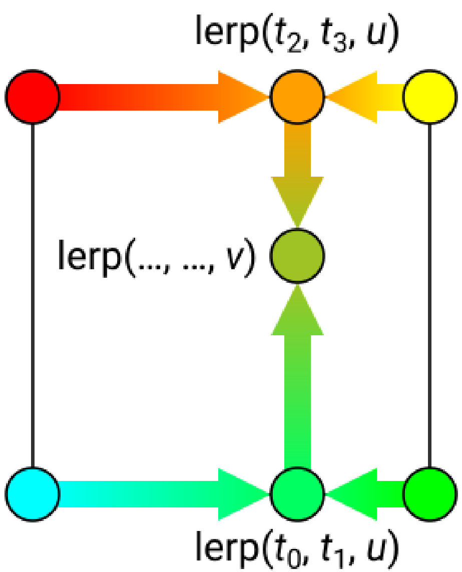
$$u_0 = \text{lerp}(s, u_{00}, u_{10})$$

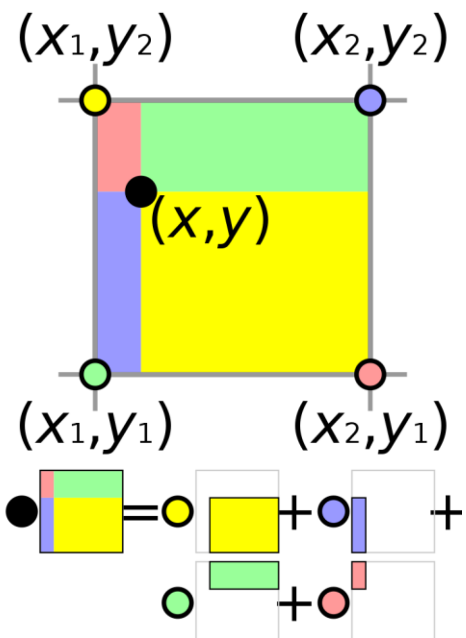
$$u_1 = \text{lerp}(s, u_{01}, u_{11})$$

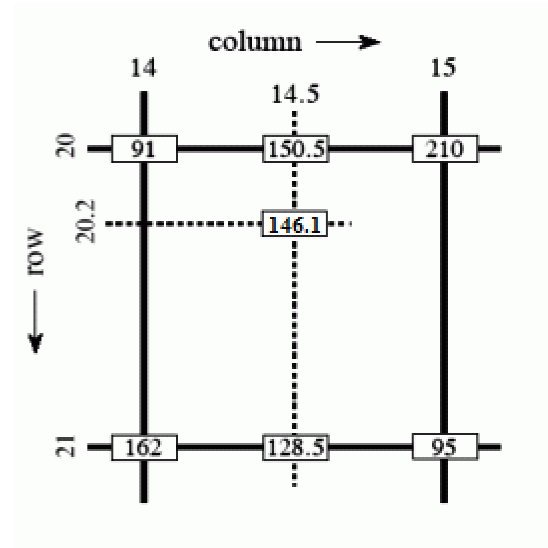
Final vertical lerp, to get result:

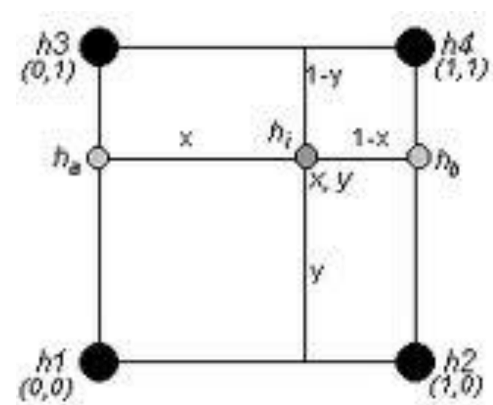
$$f(x, y) = \text{lerp}(t, u_0, u_1)$$

Bilinear interpolation

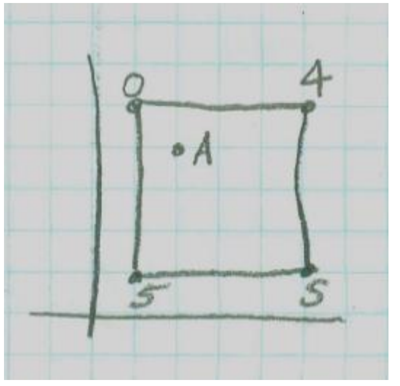








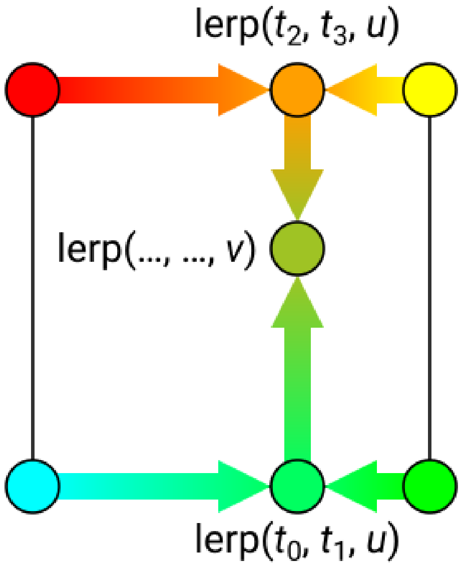
Q about bi-linear interpolation



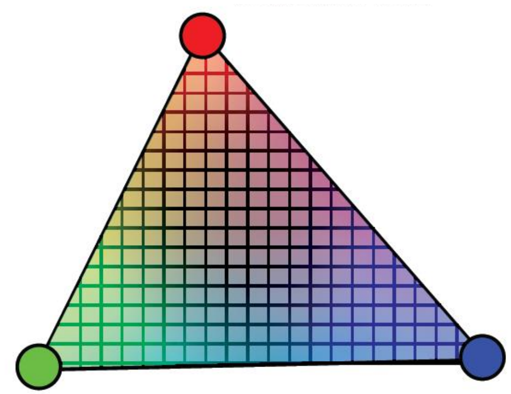
What is the value at point A?

(A) 1
(B) 2
(C) 3
(D) 4
(E) Don’t know

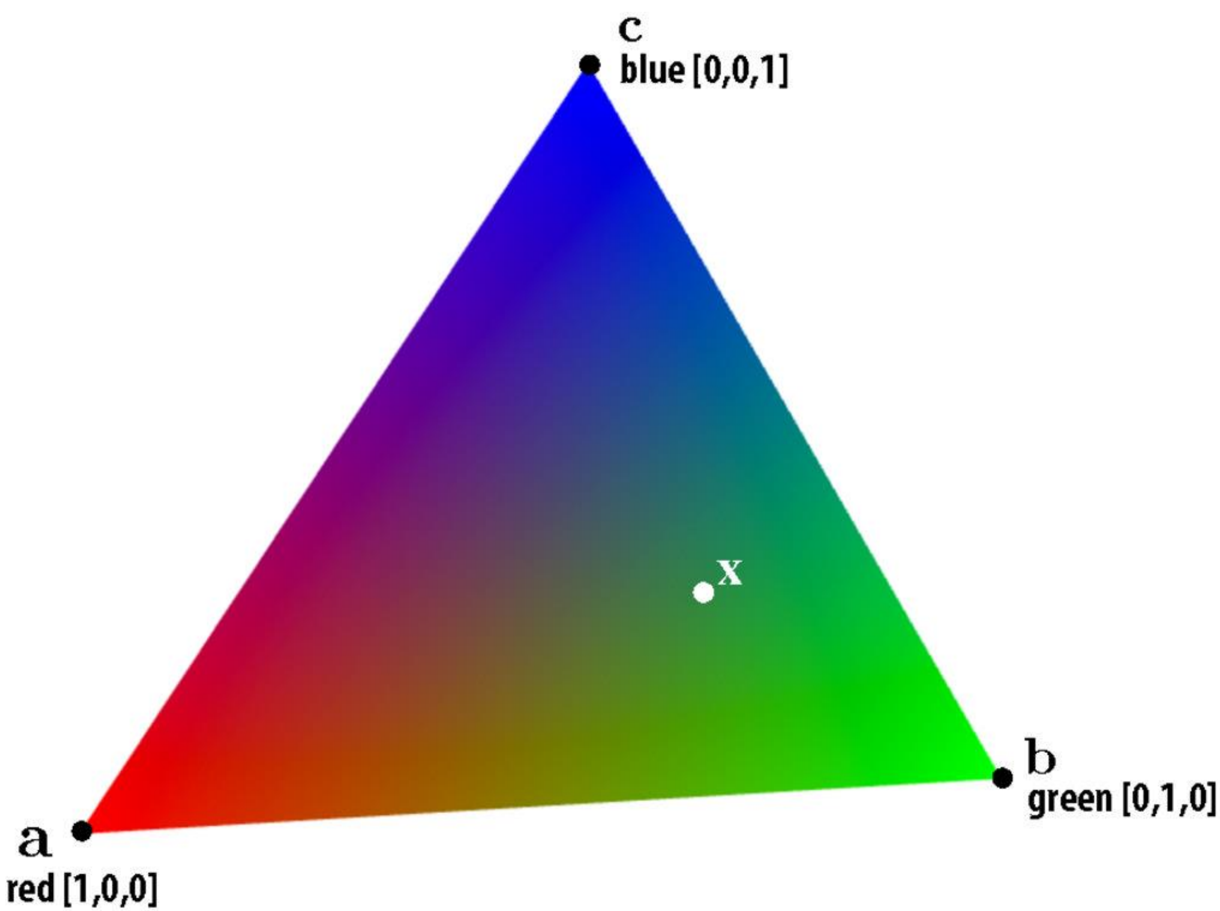
Umm.. So how do I use this on triangles?



Bilinear interpolation

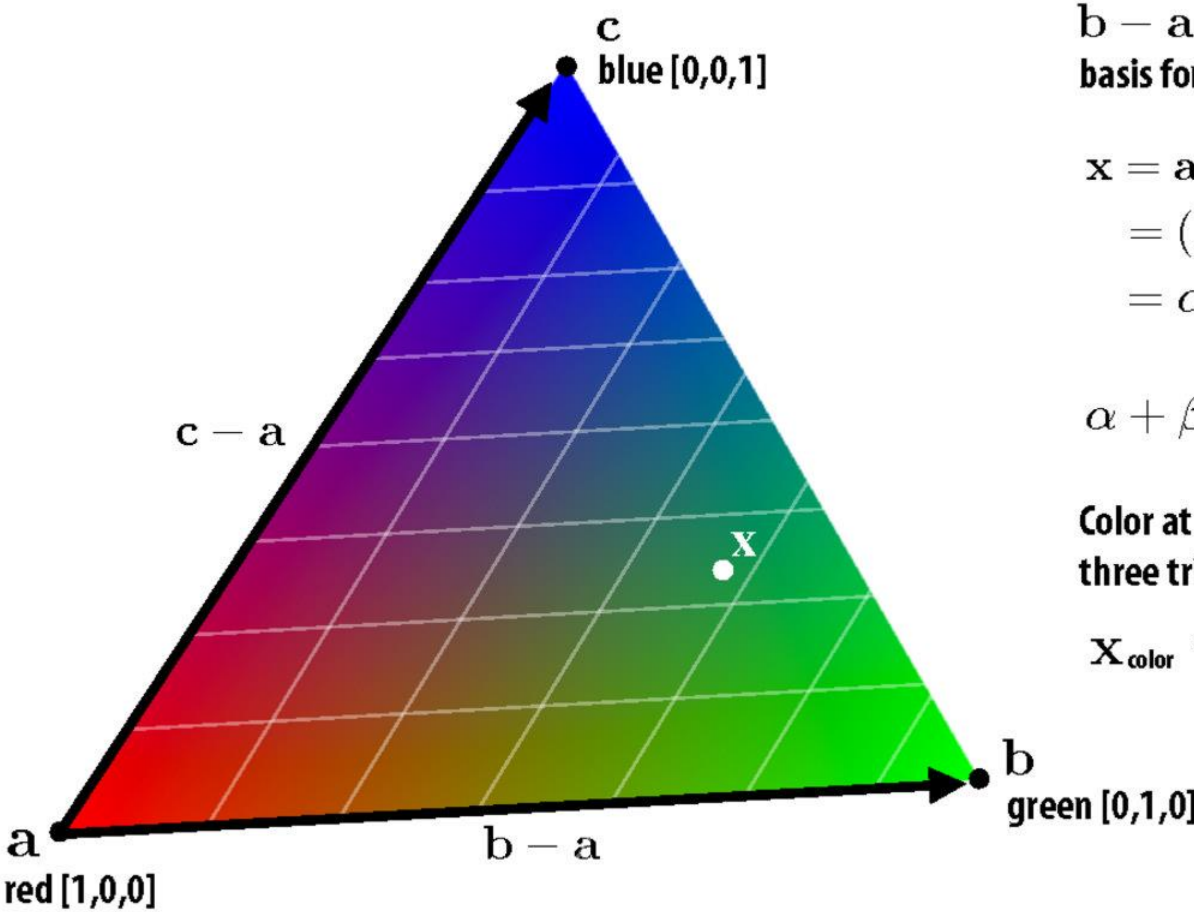


Consider sampling color(x,y)



What is the triangle's color at the point $\mathbf{x}$?

Interpolation via barycentric coordinates



$\mathbf{b} - \mathbf{a}$ and $\mathbf{c} - \mathbf{a}$ form a non-orthogonal basis for points in triangle (origin at $\mathbf{a}$)

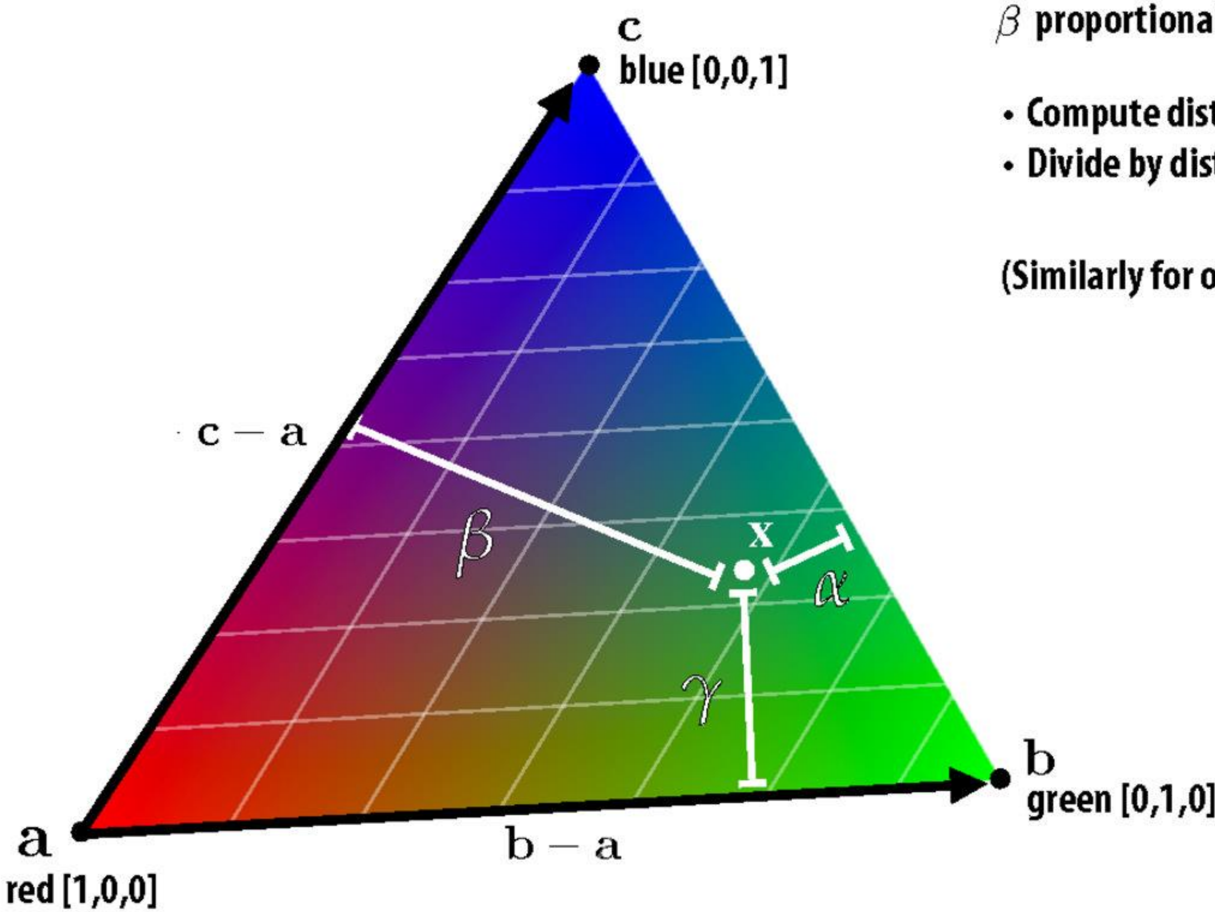
$$\begin{aligned}\mathbf{x} &= \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \\ &= (1 - \beta - \gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\ &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}\end{aligned}$$

$$\alpha + \beta + \gamma = 1$$

Color at $\mathbf{x}$ is linear combination of color at three triangle vertices.

$$\mathbf{x}_{\text{color}} = \alpha\,\mathbf{a}_{\text{color}} + \beta\,\mathbf{b}_{\text{color}} + \gamma\,\mathbf{c}_{\text{color}}$$

Barycentric coordinates as scaled distances

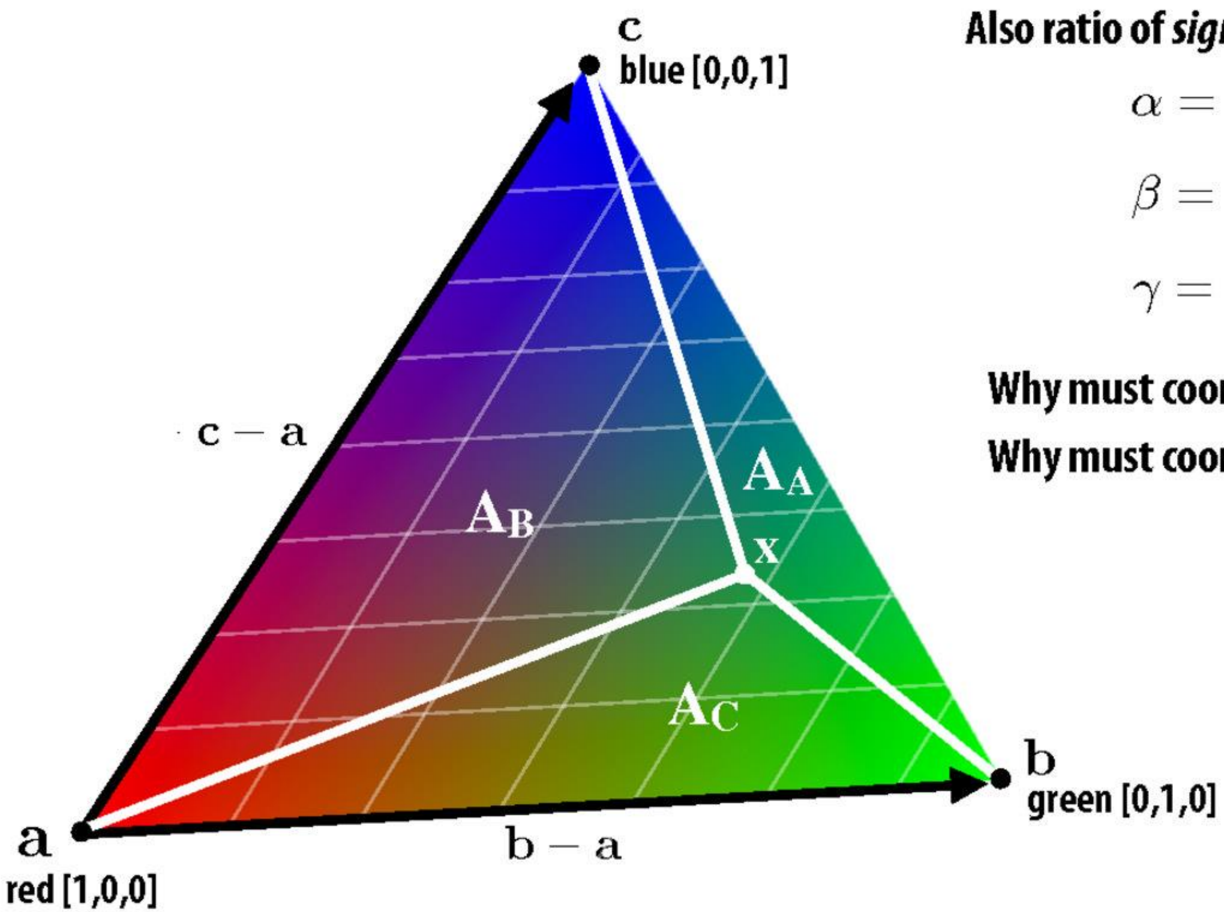


β proportional to distance from $\mathbf{x}$ to edge $\mathbf{c} - \mathbf{a}$

- Compute distance of x from line ca
- Divide by distance of b from line ca ("height")

(Similarly for other two barycentric coordinates)

Barycentric coordinates as ratio of areas



Also ratio of *signed* areas:

$$\alpha = A_A/A$$

$$\beta = A_B/A$$

$$\gamma = A_C/A$$

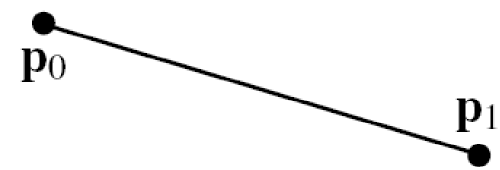
Why must coordinates sum to one?

Why must coordinates be between 0 and 1?

Barycentric Coordinates

Linear Interpolation

- Pick points along line: $\mathbf{p}(u) = (1-u)\mathbf{p}_0 + u\mathbf{p}_1$



Barycentric Coordinates

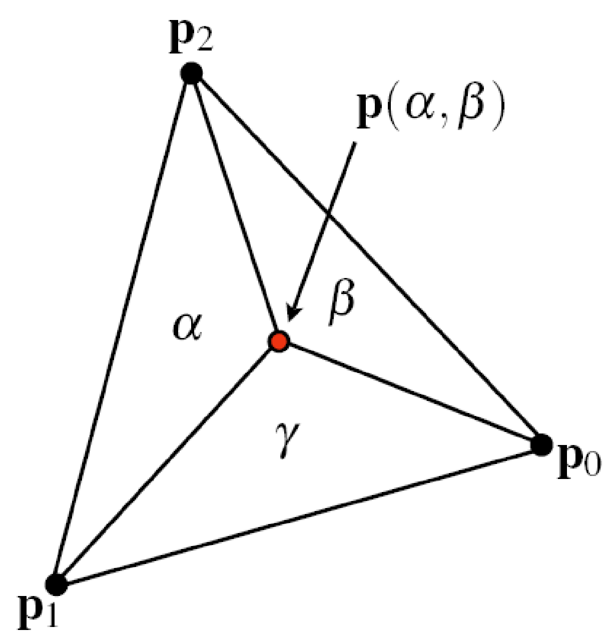
- Points in a triangle satisfy the following equation:

$$\mathbf{p} = \alpha\mathbf{p}_0 + \beta\mathbf{p}_1 + \gamma\mathbf{p}_2 \quad \text{where} \quad \alpha + \beta + \gamma = 1$$

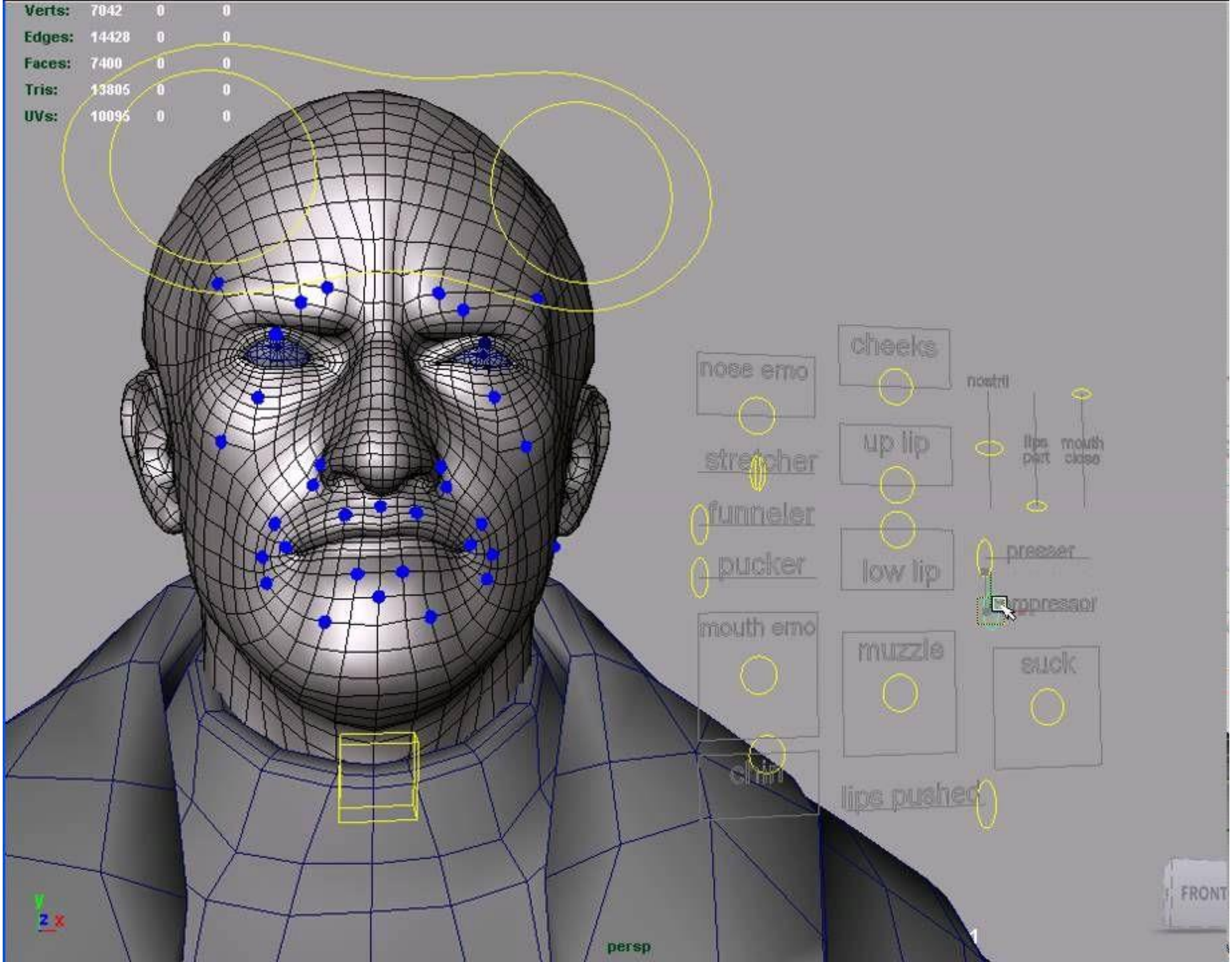
- Coefficients are area ratios:

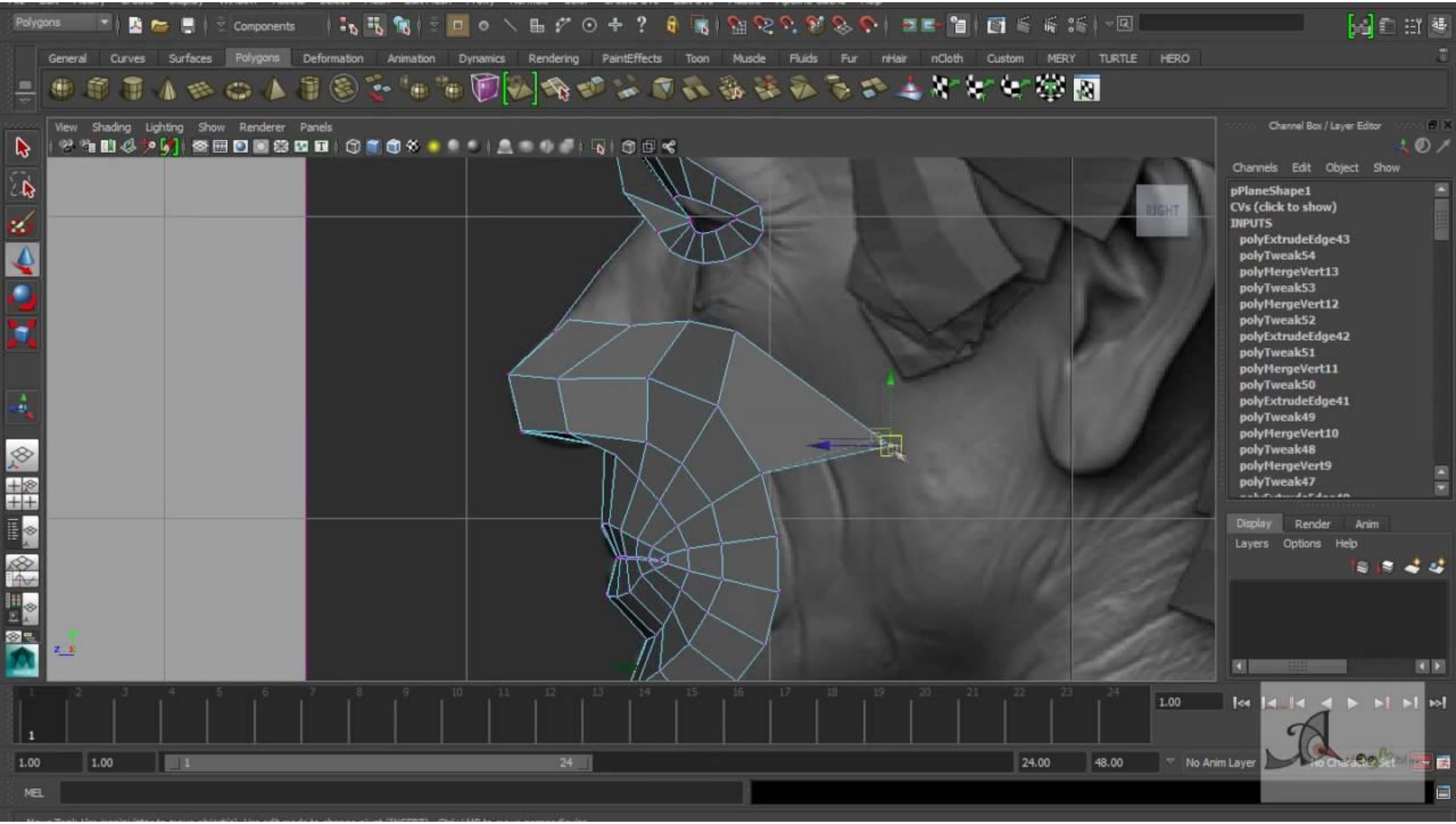
$$\alpha = \frac{Area(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p})}{Area(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)} \qquad \beta = \frac{Area(\mathbf{p}_0, \mathbf{p}_2, \mathbf{p})}{Area(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)}$$

$$\gamma = \frac{Area(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p})}{Area(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)} = 1 - \alpha - \beta$$

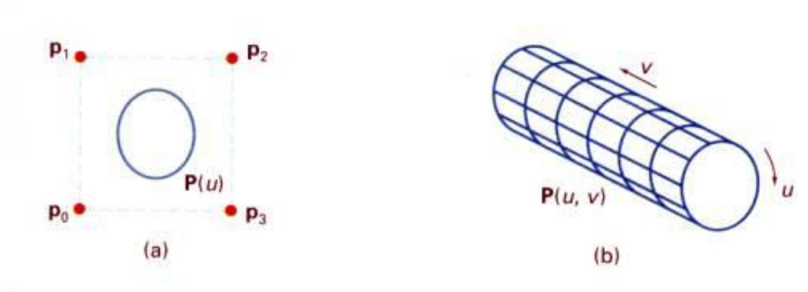


Non triangle modeling

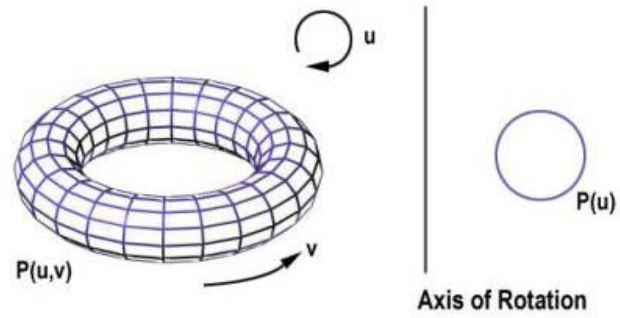

Verts: 7042 0 0
Edges: 14428 0 0
Faces: 7400 0 0
Tris: 13805 0 0
UVs: 10095 0 0
nose emo
cheeks
nostril
stretcher
up lip
lips part
mouth close
funneler
pucker
low lip
presser
mouth emo
muzzle
suck
chin
lips pushed
persp
FRONT


Polygons
Components
General
Curves
Surfaces
Polygons
Deformation
Animation
Dynamics
Rendering
PaintEffects
Toon
Muscle
Fluids
Fur
nHair
nCloth
Custom
MERY
TURTLE
HERO
View
Shading
Lighting
Show
Renderer
Panels
RIGHT
Channel Box / Layer Editor
Channels
Edit
Object
Show
pPlaneShape1
CVs (click to show)
INPUTS
polyExtrudeEdge43
polyTweak54
polyMergeVert13
polyTweak53
polyMergeVert12
polyTweak52
polyExtrudeEdge42
polyTweak51
polyMergeVert11
polyTweak50
polyExtrudeEdge41
polyTweak49
polyMergeVert10
polyTweak48
polyMergeVert9
polyTweak47
Display
Render
Anim
Layers
Options
Help
1.00
24.00
48.00
No Anim Layer
No Character Set
MEL

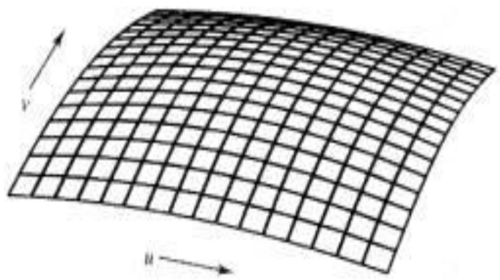
Some Non-Polygonal Modeling Tools



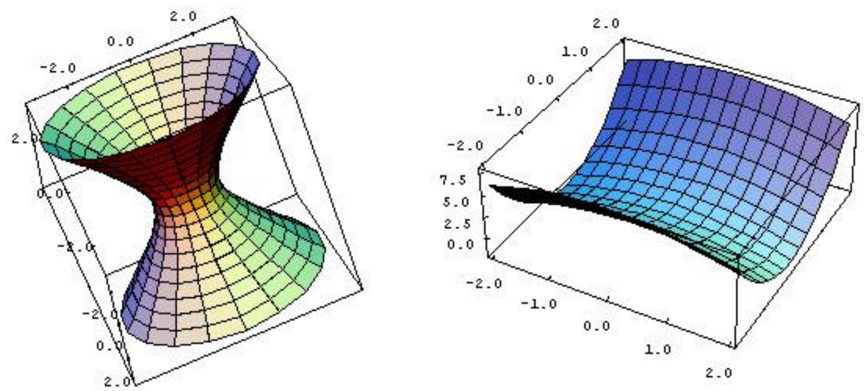
Extrusion



Surface of Revolution

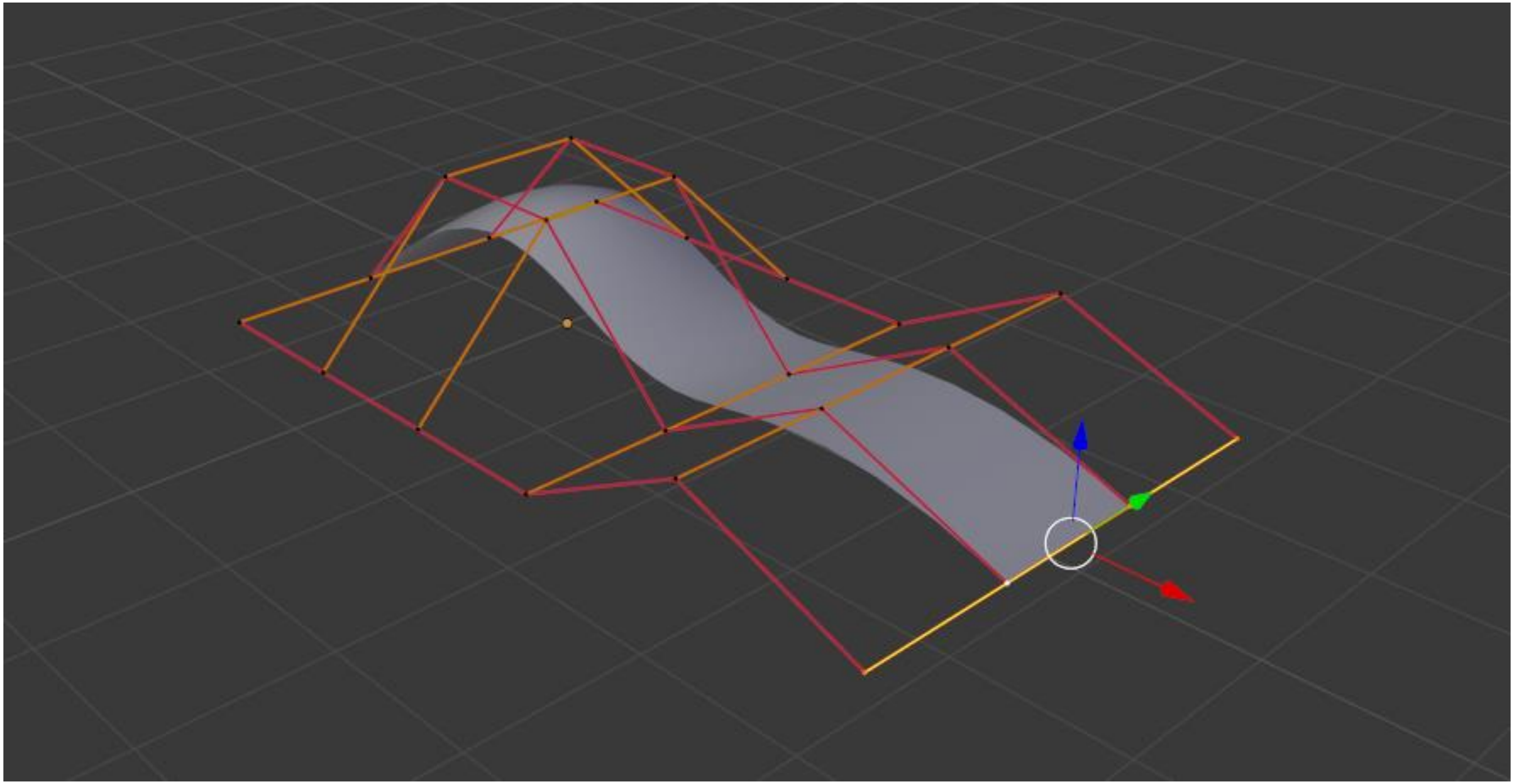


Spline Surfaces/Patches



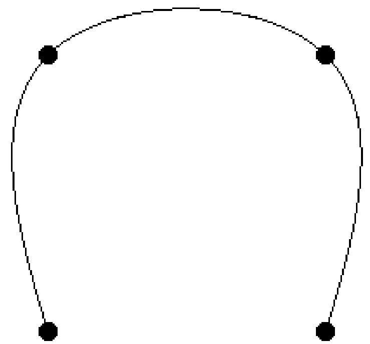
Quadrics and other implicit polynomials

Splines and patches

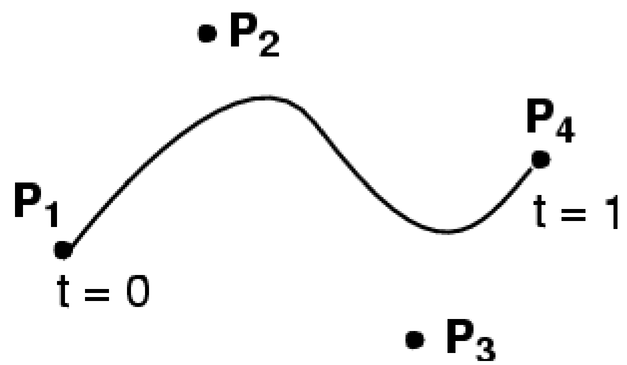


Definition: What's a Spline?

- Smooth curve defined by some control points
- Moving the control points changes the curve

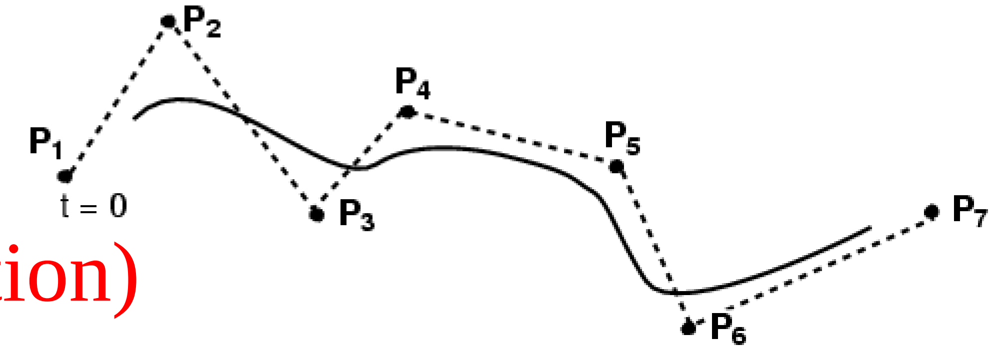


Interpolation

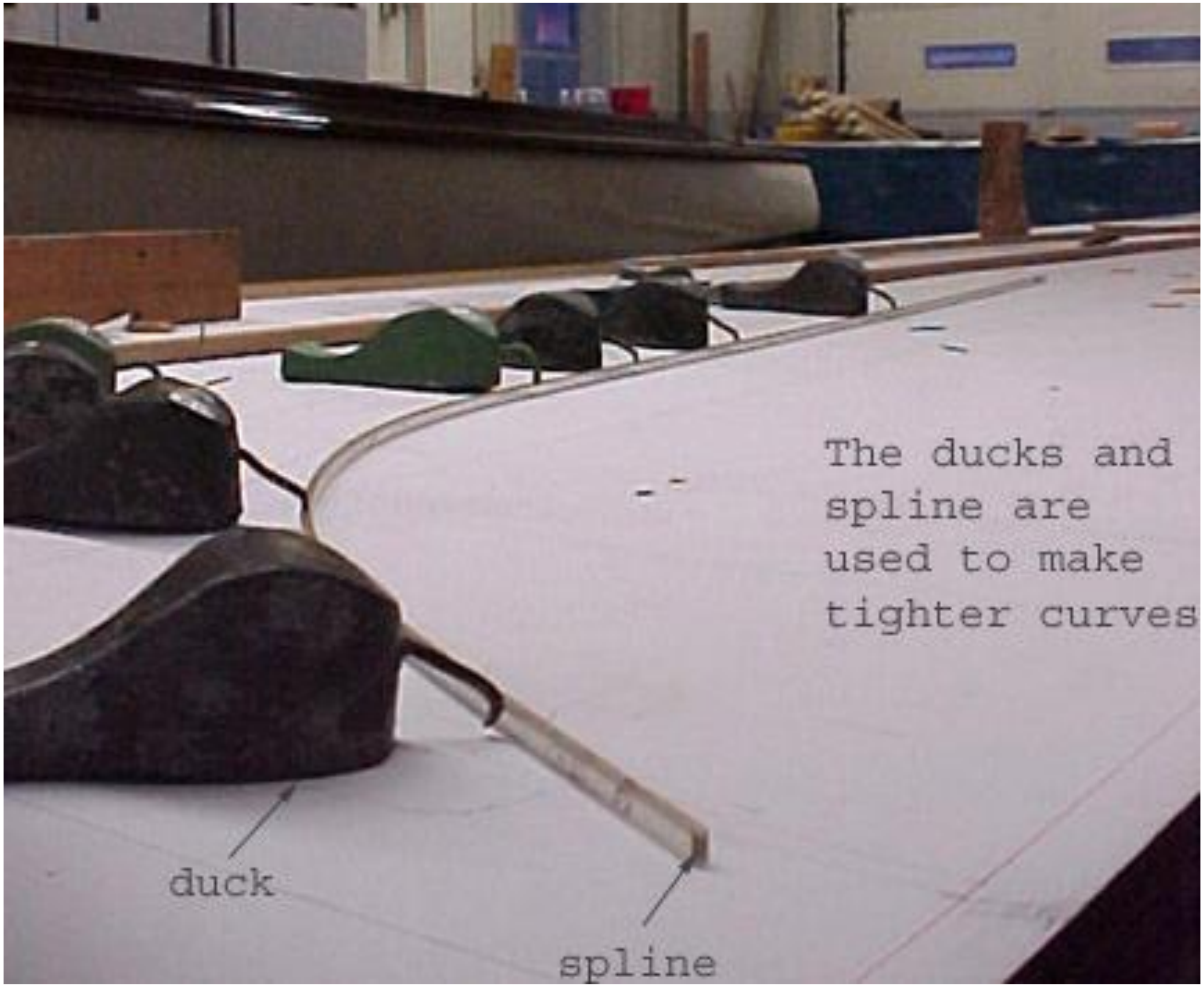


Bézier (approximation)

BSpline (approximation)



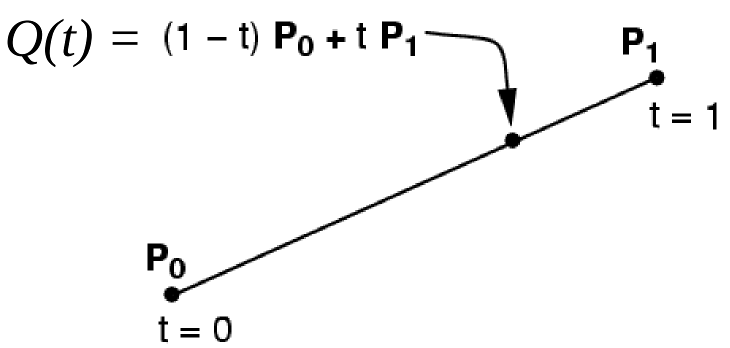
Interpolation Curves / Splines

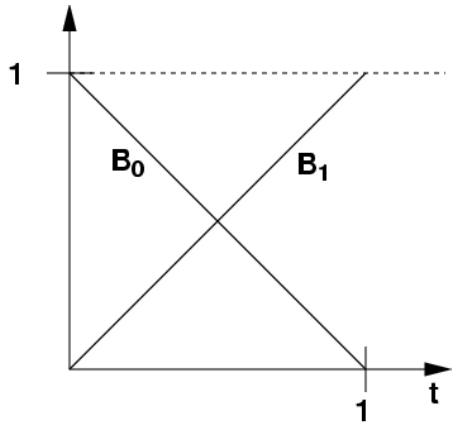


www.abm.org

Linear Interpolation

- Simplest "curve" between two points





Spline Basis Functions

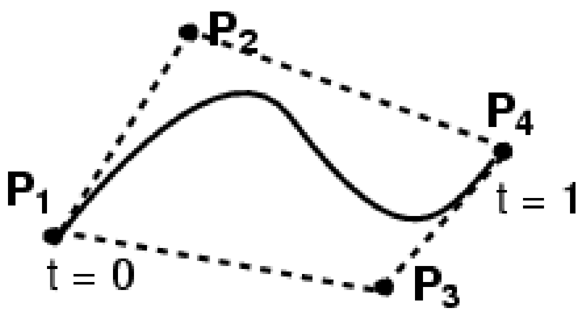
a.k.a. Blending Functions

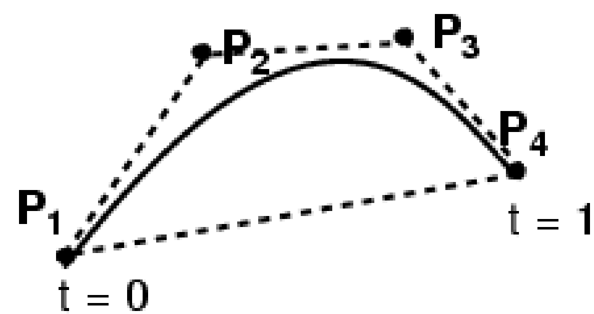
$$Q(t) = \begin{pmatrix} Q_x(t) \\ Q_y(t) \\ Q_z(t) \end{pmatrix} = \begin{pmatrix} (P_0) & (P_1) \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} t \\ 1 \end{pmatrix}$$

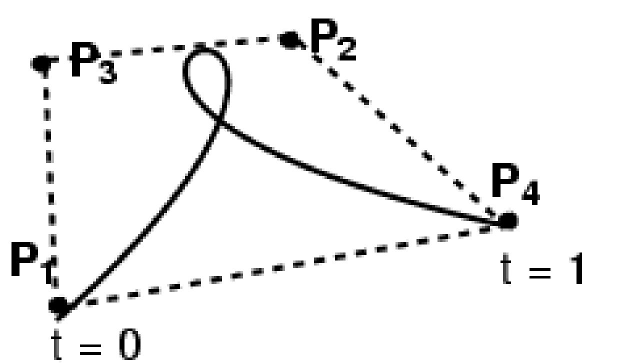
$$Q(t) = \mathbf{GBT(t)} \;\; = \text{Geometry } \mathbf{G} \cdot \text{Spline Basis } \mathbf{B} \cdot \text{Power Basis } \mathbf{T(t)}$$

Cubic Bézier Curve

- 4 control points
- Curve passes through first & last control point
- Curve is tangent at $\mathbf{P_0}$ to $(\mathbf{P_0}\text{-}\mathbf{P_1})$ and at $\mathbf{P_4}$ to $(\mathbf{P_4}\text{-}\mathbf{P_3})$

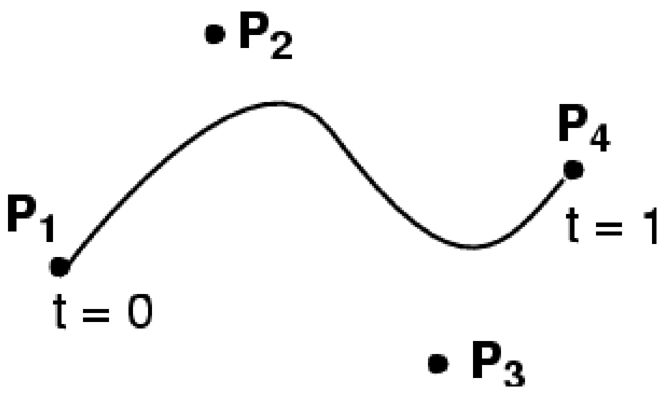


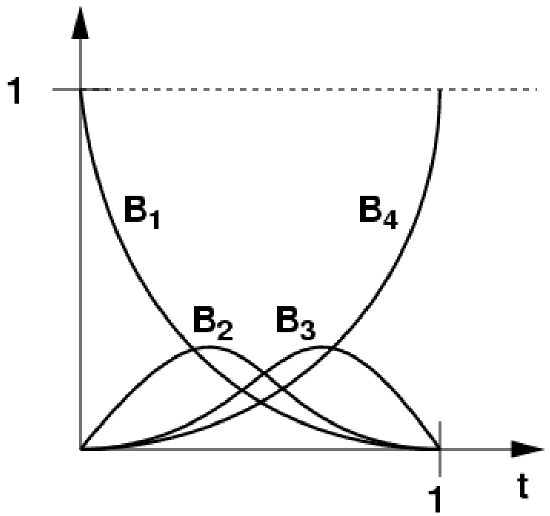




A Bézier curve is bounded by the convex hull of its control points.

Cubic Bézier Curve





$$Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4$$

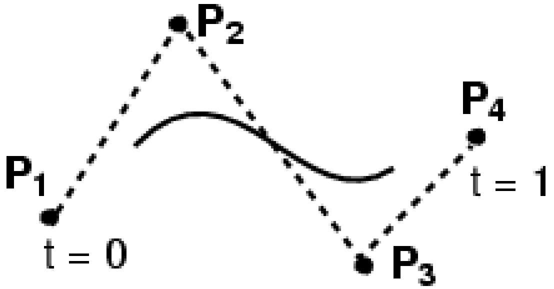
$$Q(t) = \mathbf{GBT(t)} \qquad B_{Bezier} = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

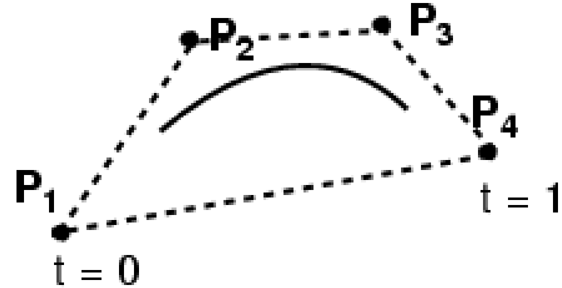
Bernstein Polynomials

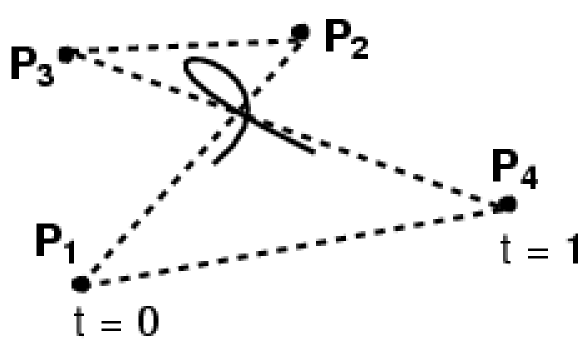
$$B_1(t) = (1-t)^3;\ B_2(t) = 3t(1-t)^2;\ B_3(t) = 3t^2(1-t);\ B_4(t) = t^3$$

Cubic BSplines

- $\geq$ 4 control points
- Locally cubic
- Curve is not constrained to pass through any control points

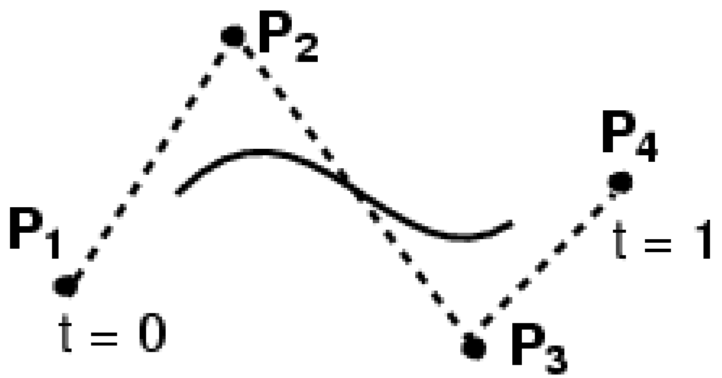


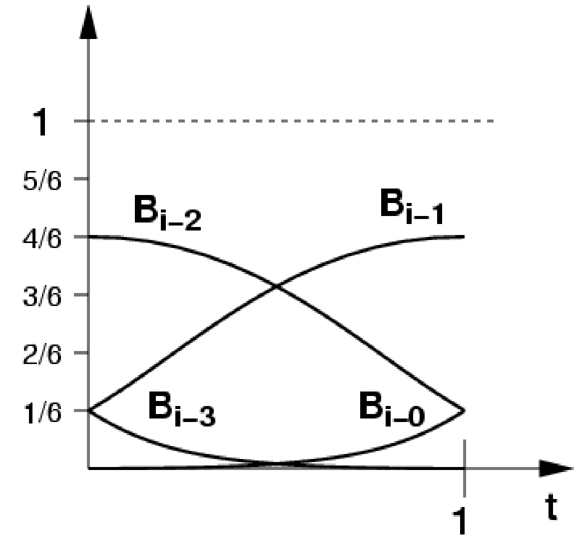




A BSpline curve is also bounded by the convex hull of its control points.

Cubic BSplines



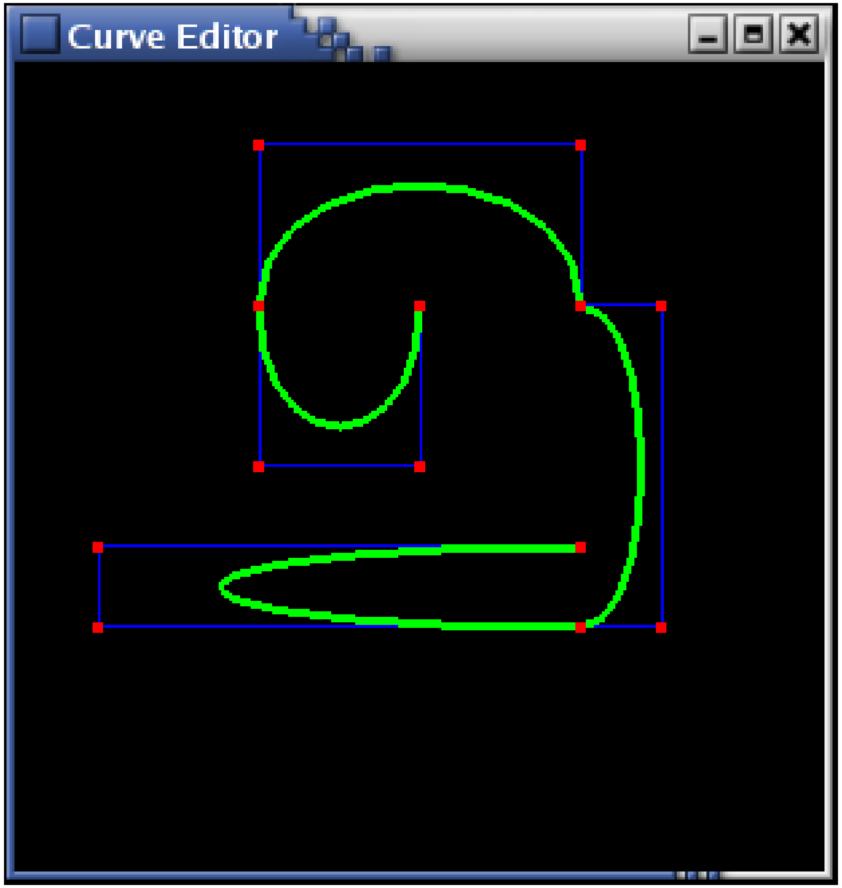


$$Q(t) = \frac{(1-t)^3}{6} P_{i-3} + \frac{3t^3 - 6t^2 + 4}{6} P_{i-2} + \frac{-3t^3 + 3t^2 + 3t + 1}{6} P_{i-1} + \frac{t^3}{6} P_i$$

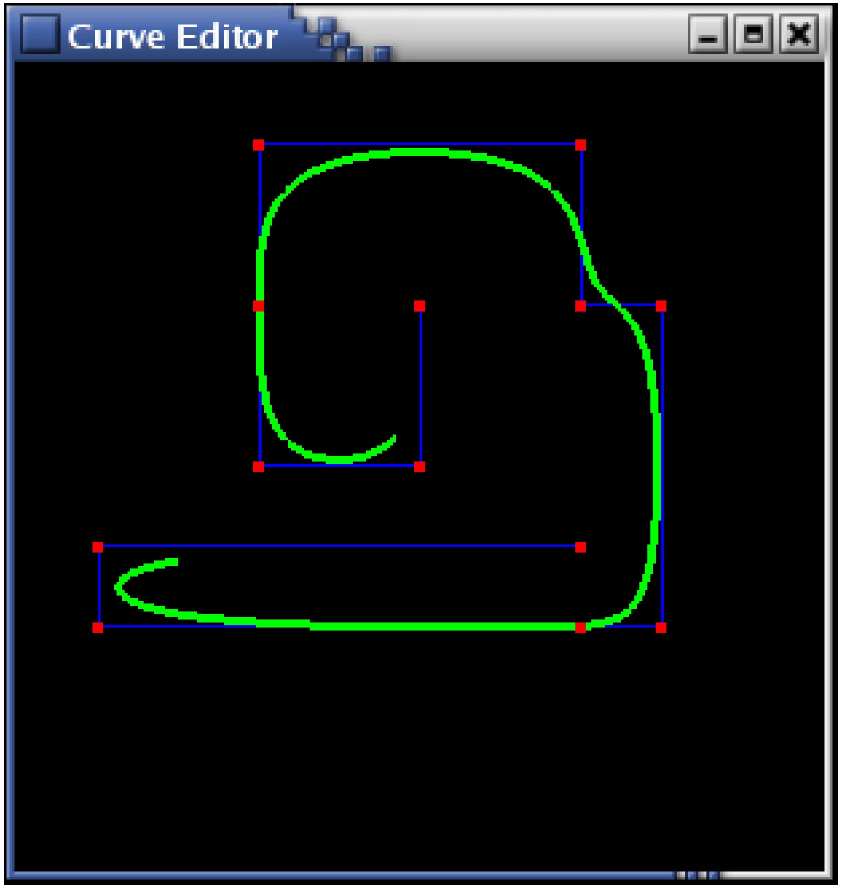
$$Q(t) = \mathbf{GBT(t)}$$

$$B_{B-Spline} = \frac{1}{6} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Bézier is not the same as BSpline



Bézier

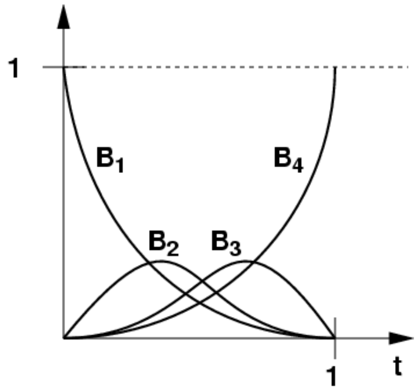


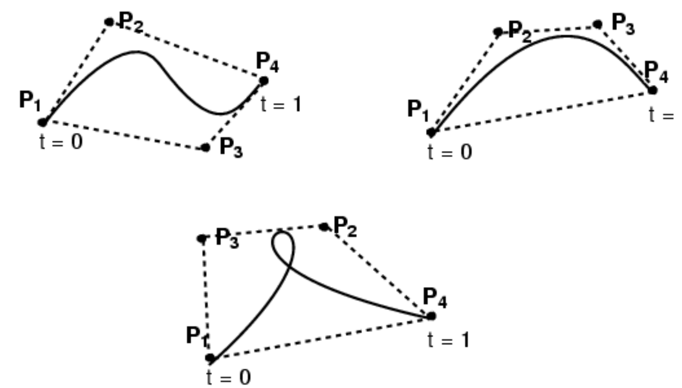
BSpline

Bézier is not the same as BSpline

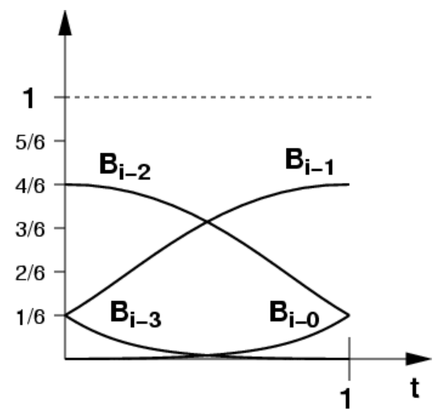
- Relationship to the control points is different

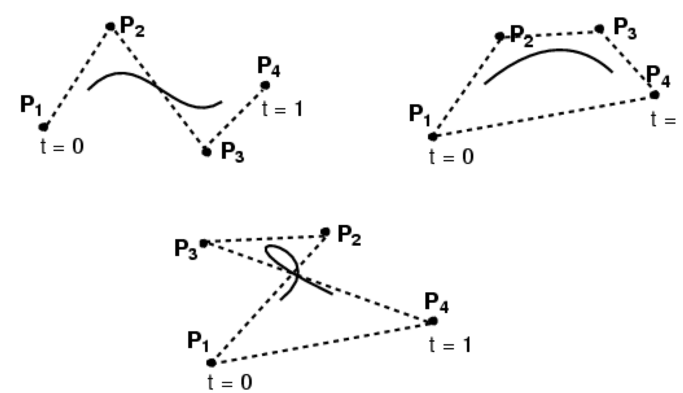
Bézier





BSpline

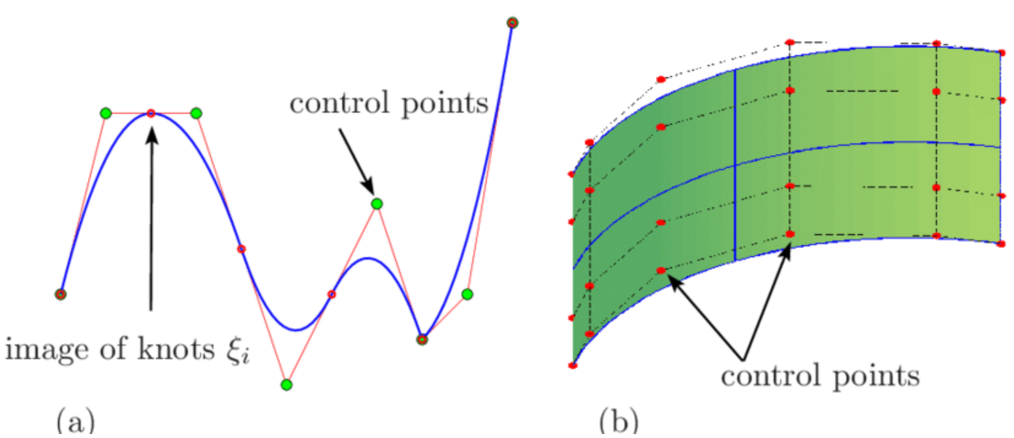




NURBS (generalized BSplines)

- BSpline: uniform cubic BSpline

- NURBS: Non-Uniform Rational BSpline
  - non-uniform = different spacing between the blending functions, a.k.a. knots
  - rational = ratio of polynomials (instead of cubic)

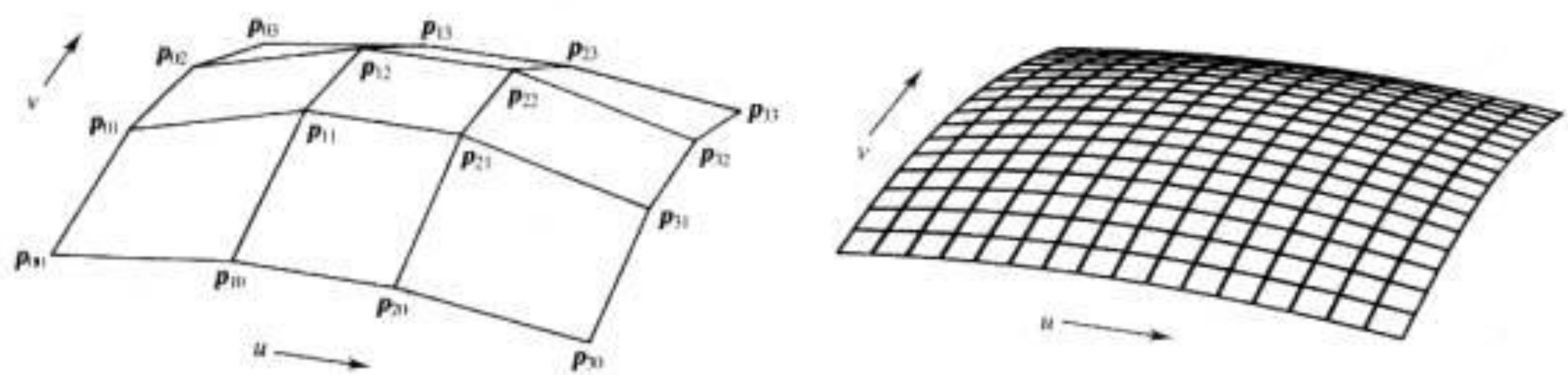


Bicubic Bezier Patch

Notation: $\mathbf{CB}(P_1, P_2, P_3, P_4, \alpha)$ is Bézier curve with control points P_i evaluated at α

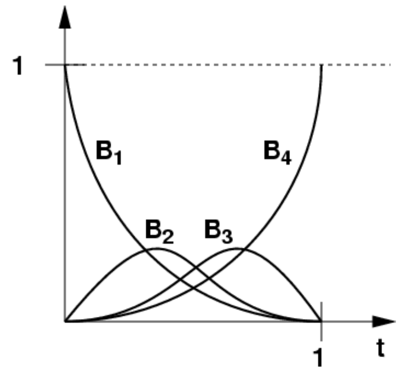
Define "Tensor-product" Bézier surface

$$\begin{aligned} Q(s,t) = \mathbf{CB}(\quad & \mathbf{CB}(P_{00}, P_{01}, P_{02}, P_{03}, t), \\ & \mathbf{CB}(P_{10}, P_{11}, P_{12}, P_{13}, t), \\ & \mathbf{CB}(P_{20}, P_{21}, P_{22}, P_{23}, t), \\ & \mathbf{CB}(P_{30}, P_{31}, P_{32}, P_{33}, t), \\ & s) \end{aligned}$$

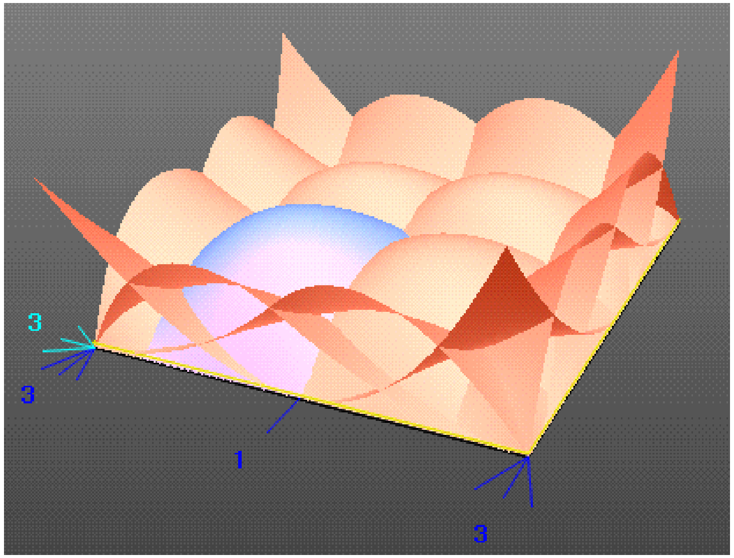


(a) (b)

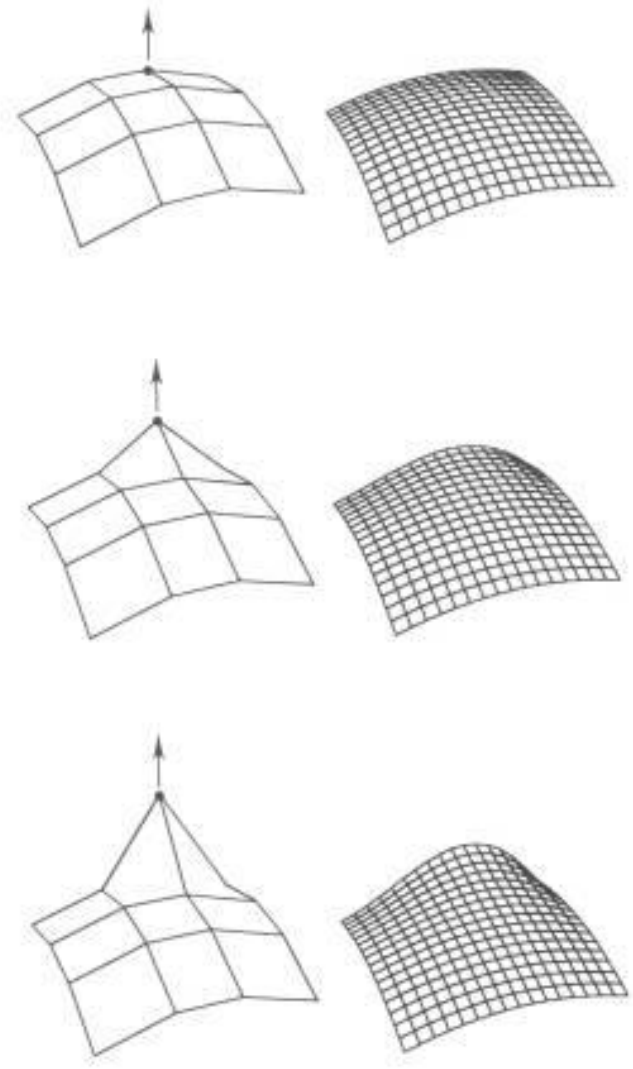
Editing Bicubic Bezier Patches



Curve Basis Functions

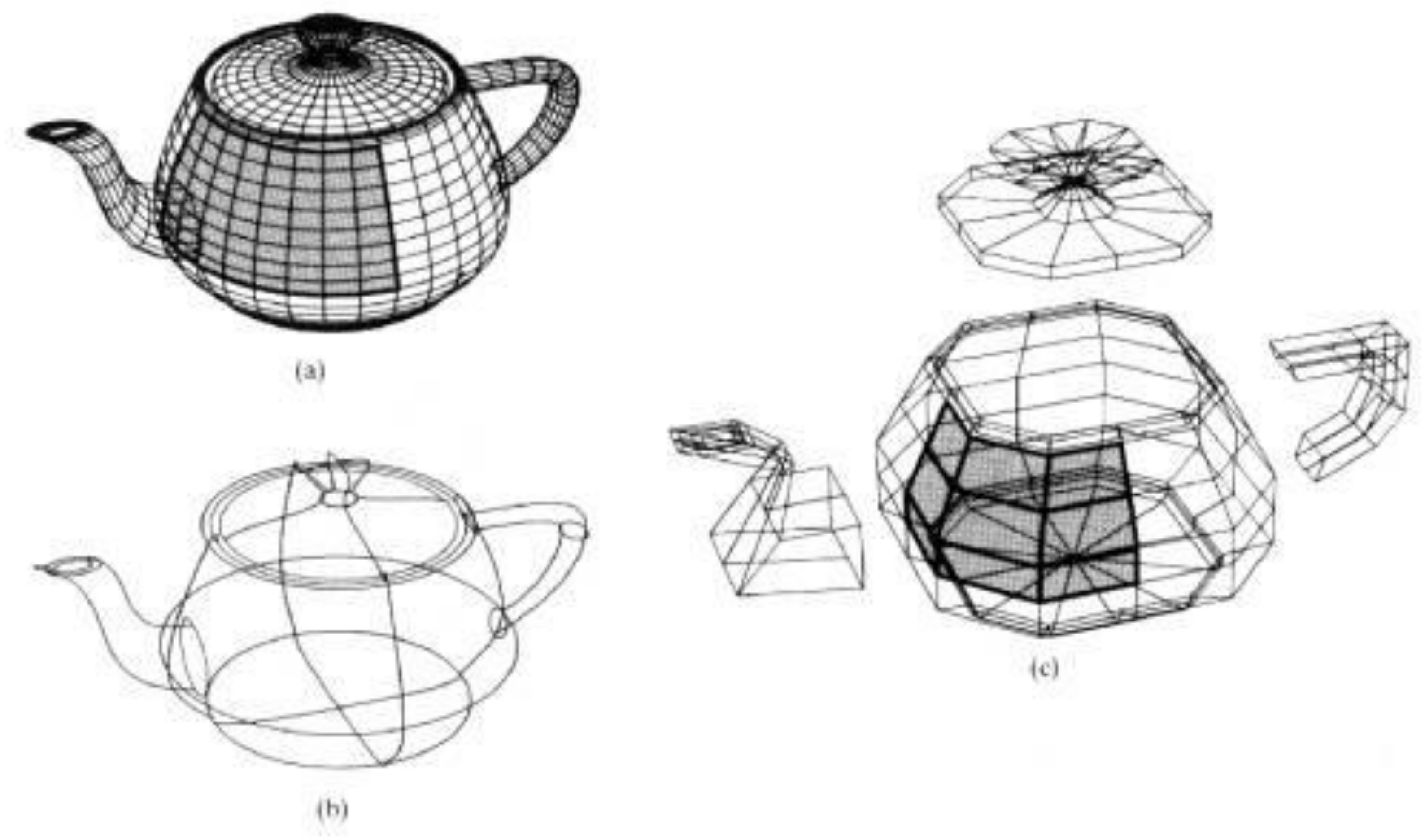


Surface Basis Functions



Modeling with Bicubic Bezier Patches

- Original Teapot specified with Bezier Patches

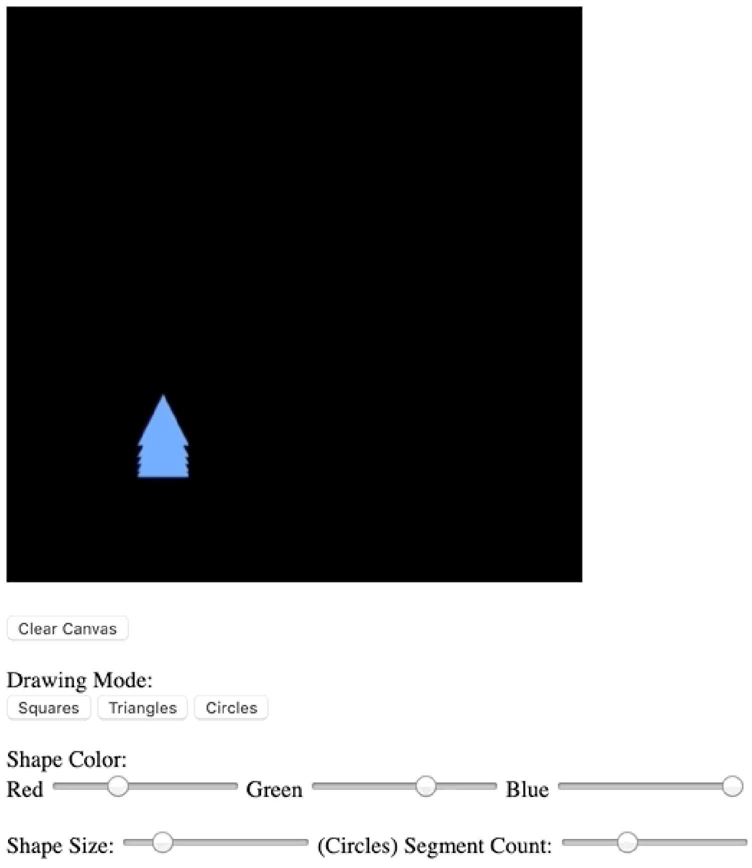


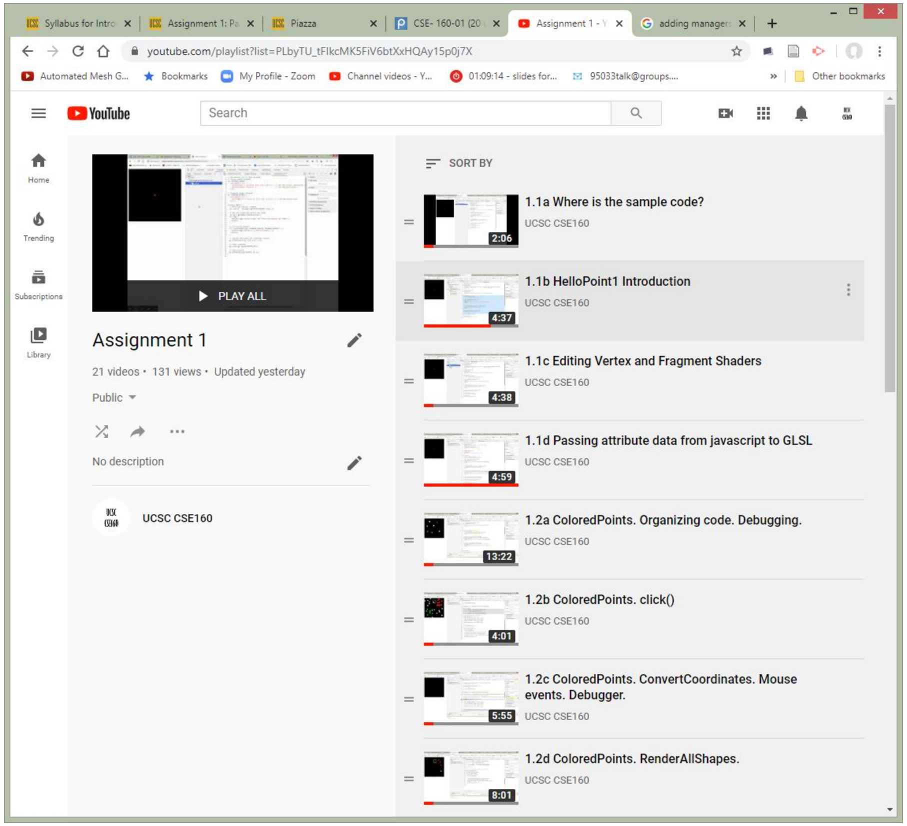
Administrative

Due Dates

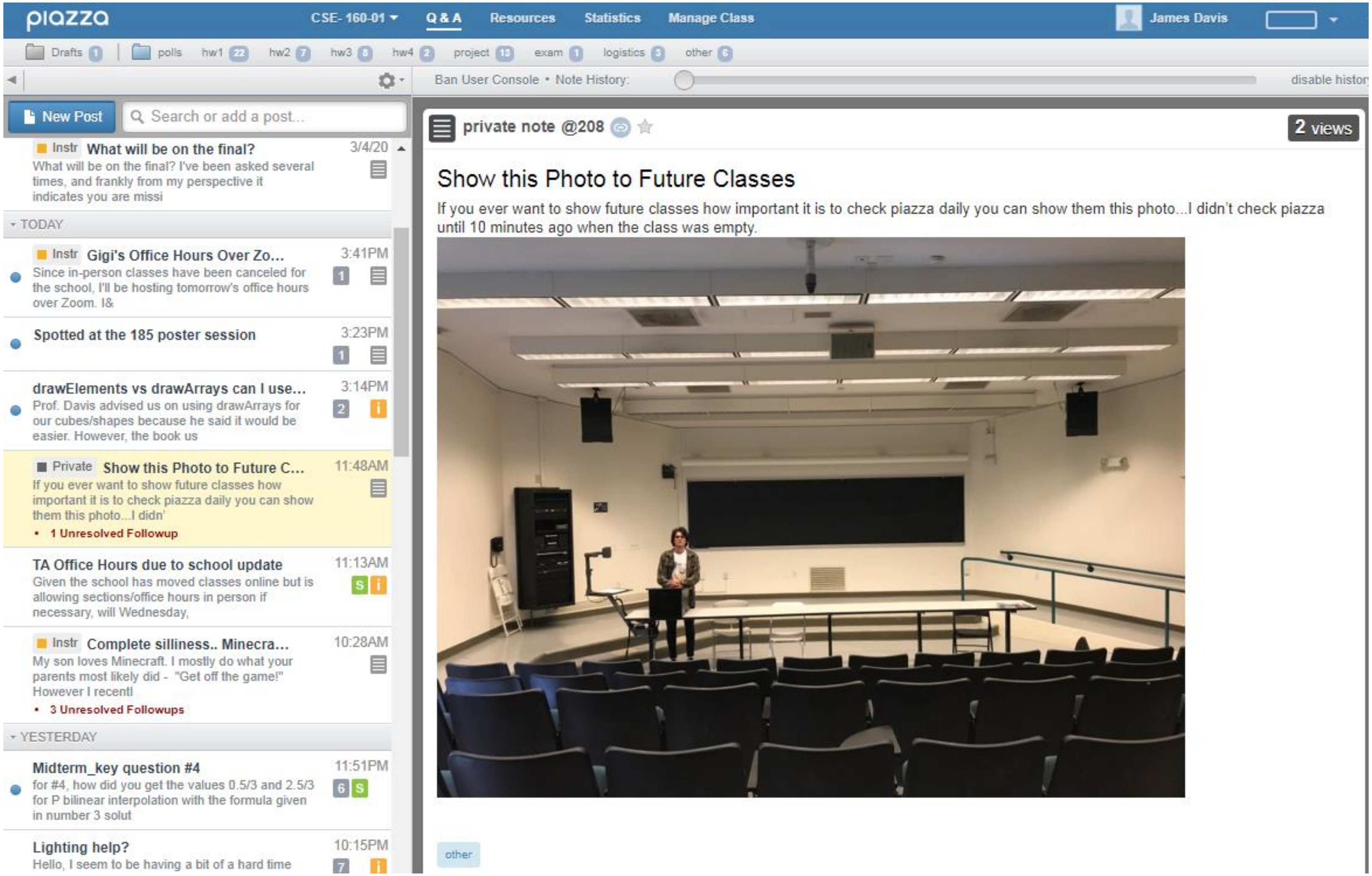
- Due Monday
  - HW 1
  - Lab Assignment 0
- Tuesday: NO ZOOM (watch videos)
- Due Wed
  - Quiz 1 (open until Wed)
- Due the following Monday
  - Assignment 1 (Paint Program)

Assignment 1 (watch videos)





Check Piazza for Announcements



[WebGL programming guide: interactive 3D graphics programming with WebGL](#)
[Kouichi Matsuda and Rodger Lea](#)

The assignments follow the flow of this book closely, and I assume you are reading the chapters. You'll want a copy.

Youtube Channel

This was new in Spring20 and students reported that it was useful, so we are keeping it. There will be playlists for different topics. e.g. Assignment1 playlist has videos to walk you through the "lab" assignment.

https://www.youtube.com/channel/UCSynd9Z5RdlpKfvTCITV_8A/playlists

Schedule

Raw lecture recordings (unedited, needs a @ucsc.edu login) Spr20

Raw Lectures Fall20

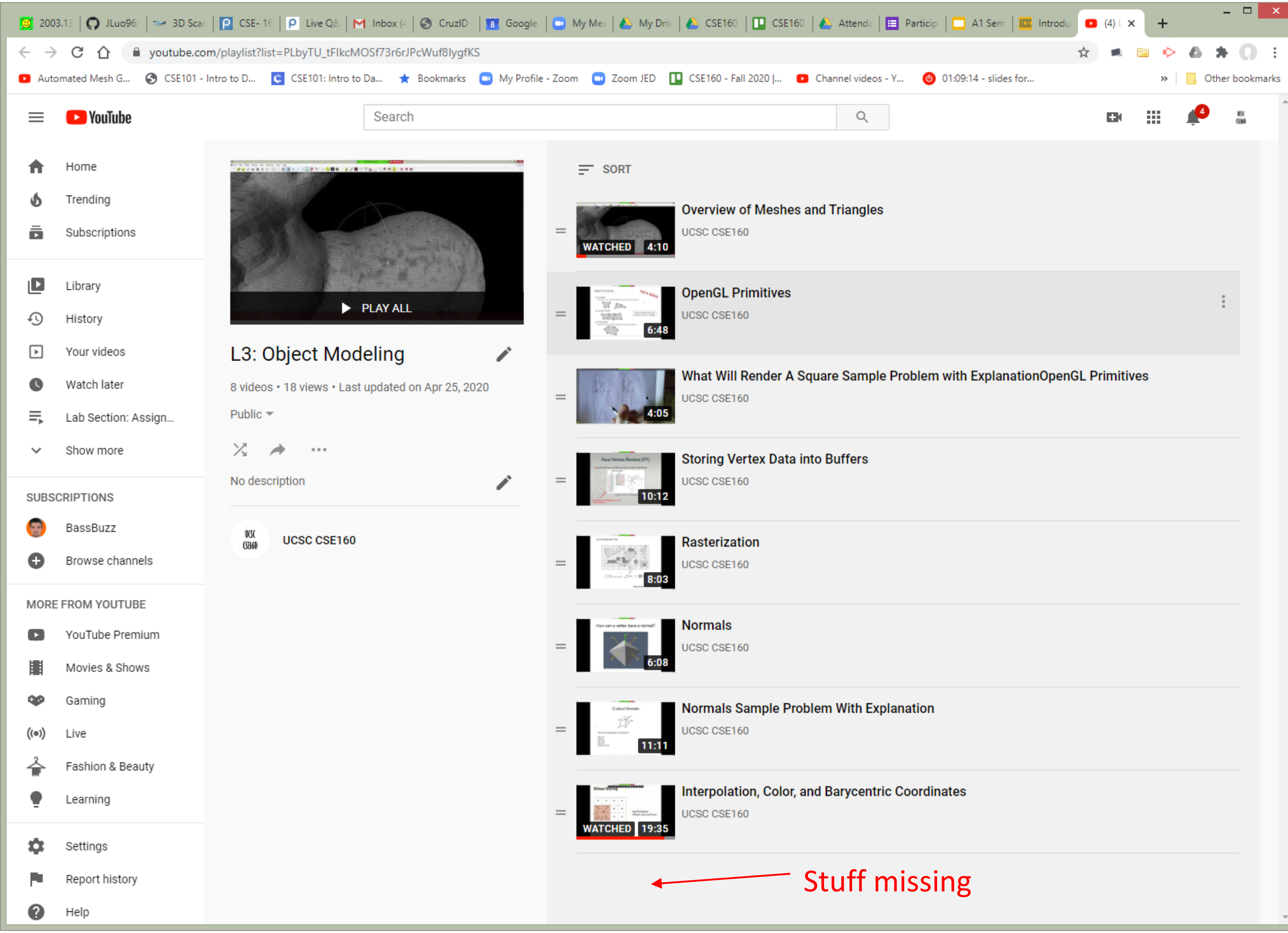
• Lecture 1 (Oct 1): Introduction to Class ◦ Slides(Fall20)	

• Lecture 2 (Oct 6): Linear Algebra Review, Intro to Shaders
  ◦ Slides(Fall20)
• Lecture 3 (Oct 8): Object Modeling, triangles and meshes.
  ◦ Videos
  ◦ Slides
  ◦ Intro to Javascript Slides
	Reading:
	• Shirley - Fundamentals of Computer Graphics - Ch 2.4-2.4.4, Ch 5.2-5.2.2)
	• Matsuda Ch 1, Ch2 (9-16 pages)
	Due: Due: Lab 0, HW1, Quiz 1 (Linear Algebra)

• Lecture 4 (Oct 13): NO ZOOM
Watch the YouTube for Asg1
• Lecture 5 (Oct 15): Transformations.
  ◦ Videos
  ◦ Slides
  ◦ transformation_game.jar
	Reading:
	• Matsuda Ch 2, Ch03 (67-91).
	Due: Lab 1 - Paint Program

• Lecture 6 (Oct 20): NO ZOOM
Watch the Youtube for Asg2.
• Lecture 7 (): Colors.
  ◦ Videos
  ◦ Slides
	Reading:
	• Shirley Ch 6 p135-158 - Transformation Matrices.
	• Matsuda Chap 3(91-113)
	Due: HW 2, Quiz 2 (ObjectModeling, Transforms)

• Lecture 8 (Oct 27): *Discuss quiz2, Non-triangle Modeling*	Reading:


youtube.com/playlist?list=PLbyTU_tFlkcMOSf73r6rJPcWuf8lygfKS
Automated Mesh G...
CSE101 - Intro to D...
CSE101: Intro to Da...
Bookmarks
My Profile - Zoom
Zoom JED
CSE160 - Fall 2020 |...
Channel videos - Y...
01:09:14 - slides for...
Other bookmarks
YouTube
Search
Home
Trending
Subscriptions
Library
History
Your videos
Watch later
Lab Section: Assign...
Show more
SUBSCRIPTIONS
BassBuzz
Browse channels
MORE FROM YOUTUBE
YouTube Premium
Movies & Shows
Gaming
Live
Fashion & Beauty
Learning
Settings
Report history
Help
PLAY ALL
L3: Object Modeling
8 videos • 18 views • Last updated on Apr 25, 2020
Public
No description
UCSC CSE160
SORT
WATCHED 4:10
Overview of Meshes and Triangles
UCSC CSE160
6:48
OpenGL Primitives
UCSC CSE160
4:05
What Will Render A Square Sample Problem with ExplanationOpenGL Primitives
UCSC CSE160
10:12
Storing Vertex Data into Buffers
UCSC CSE160
8:03
Rasterization
UCSC CSE160
6:08
Normals
UCSC CSE160
11:11
Normals Sample Problem With Explanation
UCSC CSE160
WATCHED 19:35
Interpolation, Color, and Barycentric Coordinates
UCSC CSE160
Stuff missing

Q&A

End