Lecture 2

Math Review 2
Introduction to OpenGL

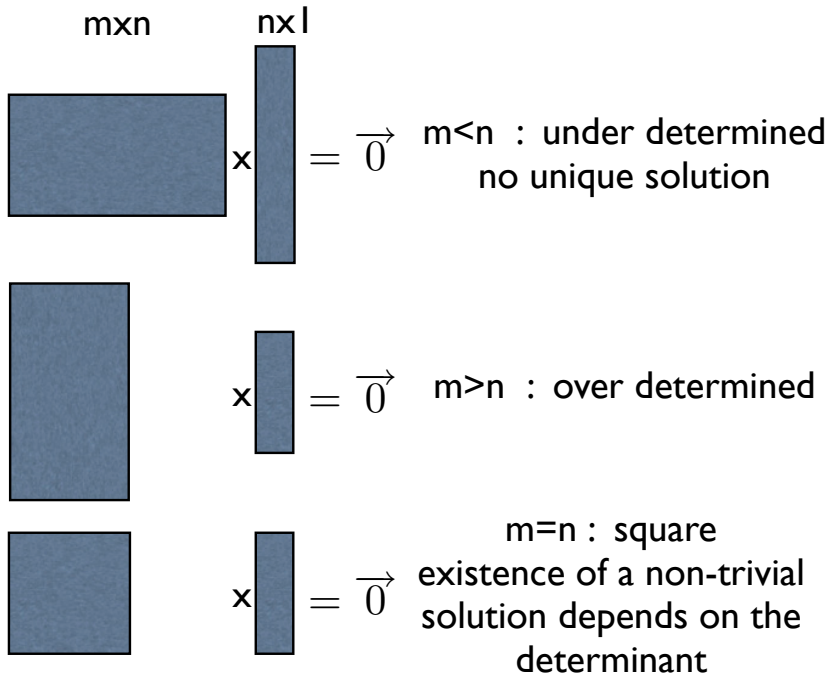
$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$Ax = 0$$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= 0 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= 0 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n &= 0 \end{aligned}$$

Determinants

- A scalar assigned to a square matrix, a measure
- Useful in analysis and solution of systems of equations
- Each matrix ==> system of equations



For a square matrix

- Non-trivial solution to Ax = 0 exists iff det = 0
- Has an inverse iff det != 0
- det = 0 : singular

Computation of Determinant

Cofactor Expansion : write an nxn determinant in terms of (n-1)x(n-1) determinants → Minors and Cofactors

Minor M_{ij} = (n-1)x(n-1) submatrix acquired by removing row i, column j.

Cofactor $k_{ij} = (-1)^{i+j} det(M_{ij})$

Row Cofactor Theorem :
For any row "i" of an nxn matrix A

$$det(A) = \sum_{j=1}^{n} a_{ij} k_{ij}$$

Computation of Determinant

Cofactor Expansion : write an nxn determinant in terms of (n-1)x(n-1) determinants → Minors and Cofactors

Minor M_{ij} = (n-1)x(n-1) submatrix acquired by removing row i, column j.

Cofactor $k_{ij} = (-1)^{i+j} det(M_{ij})$

$$\begin{bmatrix} + & - & + & - & + \\ - & + & - & + & - \\ + & - & + & - & + \\ - & + & - & + & - \\ + & - & + & - & + \end{bmatrix}$$

Computation of Determinant

- 1x1 $|a_{11}| = a_{11}$
- 2x2 $\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$
- 3x3 $\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$

$$= a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{33} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

$$= a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{23}a_{31}) + a_{13}(a_{21}a_{32} - a_{22}a_{31})$$

Back to the Cross Product...

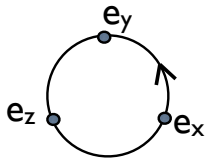
$$v \times w = (v_x e_x + v_y e_y + v_z e_z) \times (w_x e_x + w_y e_y + w_z e_z)$$

$$= \ldots$$

$$= (v_y w_z - v_z w_y) e_x + (v_z w_x - v_x w_z) e_y + (v_x w_y - v_y w_x) e_z$$

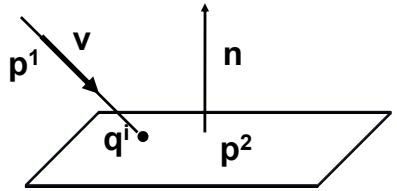
$$= \begin{vmatrix} e_x & e_y & e_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{vmatrix}$$

right handed coord sys



Intersecting a line and a plane

Same old trick: use the parametric equation for the line, implicit for the plane.



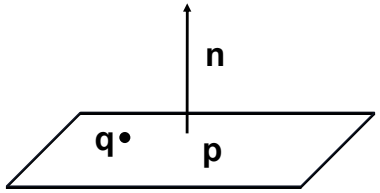
$$(p^1 + vt^i - p^2 \cdot n) = 0$$

$$t^i = -\frac{(p^1 - p^2 \cdot n)}{(v \cdot n)}$$

Do not forget to check for zero in the denominator!

Plane equations

implicit equation: (q-p,n)=0, exactly like line in 2D!



parametric equation: 2 parameters t_1,t_2

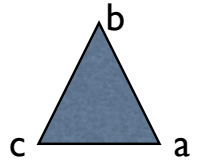
$q(t_1,t_2) = v_1 t_1 + v_2 t_2$, where v_1 and v_2 are two vectors in the plane.

$$v_1 \times v_2 = n$$

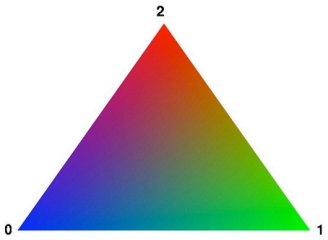
Triangles

- fundamental modeling primitives
- area in 2D

$$A = \frac{1}{2} \begin{vmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \end{vmatrix}$$

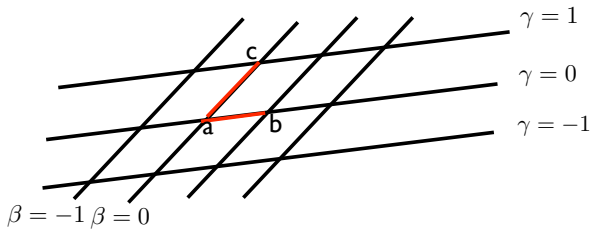


- information per vertex and interpolation



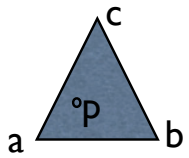
Triangles

- non-orthogonal coordinate system :



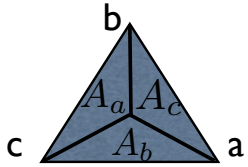
- any point represented by (β, γ) : p = a + γ(c-a) + β(b-a)
- barycentric (barycenter = center of mass)

$$p = (1 - \beta - \gamma)a + \beta b + \gamma c$$



Triangles

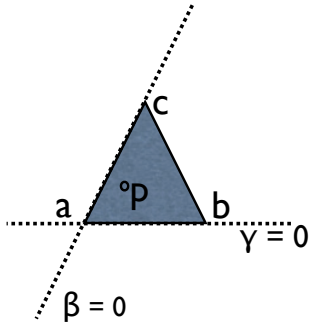
- computation
  - solve a linear system based on p = a + γ(c-a) + β(b-a)
  - or:



$$\gamma = A_a/A$$
$$\beta = A_b/A$$
$$1 - \gamma - \beta = A_c/A$$

Triangles

$$p = (1 - \beta - \gamma)a + \beta b + \gamma c$$



- inside/outside/on edge trivial
  - inside : 0 < β, γ, 1-γ-β < 1
  - on edge : either one of β, γ, 1-γ-β = 0
  - on vertex : two of β, γ, 1-γ-β = 0

Triangles - 3D

- same formula holds:

$$p = (1 - \beta - \gamma)a + \beta b + \gamma c$$

- normal $n = (b - a) \times (c - a)$
- area $(1/2)||n||$

!! $\gamma = \dfrac{n \cdot n_a}{||n||^2}$ $\beta = \dfrac{n \cdot n_b}{||n||^2}$ $1 - \gamma - \beta = \dfrac{n \cdot n_c}{||n||^2}$

Introduction to OpenGL

What is OpenGL?

- software library that creates an interface to graphics hardware
- hardware and OS independent
  - may use accelerated hardware for rendering
- low-level interface
  - polygon based rendering
  - small number of geometric primitives: pts, lines, polygons

Extensions

- OpenGL Utility Library (GLU)
  - routines implemented in terms of OpenGL commands
  - include viewing transformations, 3D models, quadric surfaces, etc...
- OpenGL Utility Toolkit (GLUT)
  - window system independent
  - simple windowing facilities for OpenGL
- X Window System extension (GLX)
- Windows extension (WGL)

OpenGL : A State Machine

- As a primitive is drawn each vertex is affected by current opengl states
- States stay in effect until changed.
- All states have a default value

OpenGL : A State Machine

Types of States

On/Off States

- glEnable(...), glDisable(...), glIsEnabled(...)
- E.g: GL_POINT_SMOOTH, GL_LINE_STIPPLE, ...

Mode States

- State Values may be one of options
- E.g. glShadeModel(GL_SMOOTH), glShadeModel(GL_FLAT)
- Query by: glGet(GL_SHADE_MODEL)

OpenGL Basics

Commands :

prefix "**gl**" and initial capital letters for each word

e.g. glClearColor()

Constants:

begin with " **GL_** " , all capital, underscores for word separation

e.g. GL_COLOR_BUFFER_BIT

Replace gl & GL_ with glut & GLUT_ for glut
and with glu & GLU_ for glu

OpenGL : A State Machine

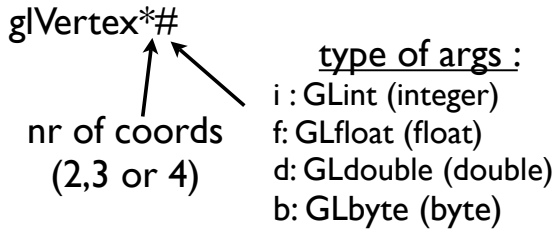
Types of States(Cont'd)

Value States

- Assign relevant values to states.
- Query values by: glGetBoolean(...), glGetDouble(...), ...
- Set values by , e.g.:
  - Color: glColor3f(GLFloat r, GLFloat g GLFloat b)
  - Point Size : glPointSize(GLFloat size);
  - Line Width : glLineWidth(GLFloat width);

OpenGL Basics

Basic Command:



...

- Same format used in other commands such as: glColor*#, glRasterPos*#, glNormal*#
- Sometimes "v" added to pass a vector(array)

OpenGL Primitives

```
glBegin(GLenum mode);
        glVertex3f(0., 0., 0.);
        glVertex3f(0., 3., 0.);
        glVertex3f(4., 3., 0.);
            ....
        glVertex3f(5., 2., 0.);
        glVertex3f(4., 0., 0.);
glEnd();
```


OpenGL Primitives

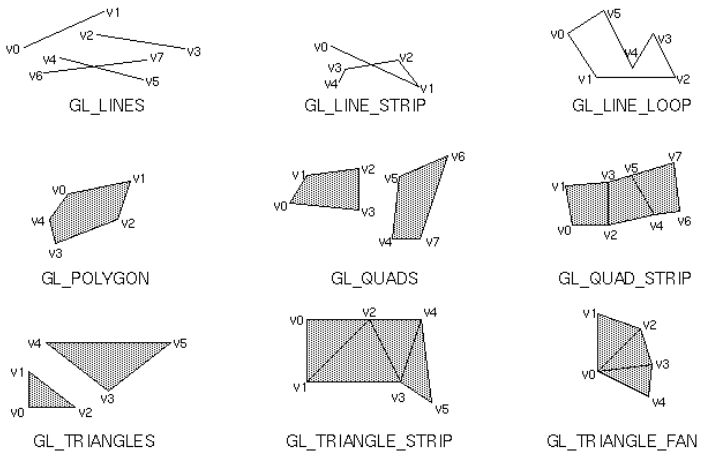
```
glBegin(GLenum mode);
        glColor(....);
        glNormal(...);
        glTexCoord(...);
        ...
        glVertex(...);

glEnd();
```

Vertex Attributes

All need to come before call to glVertex to affect that vertex

OpenGL Primitives

glBegin(GLenum mode);



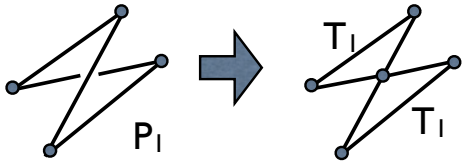
glEnd();

OpenGL Polygons

- Can be complicated and slow to draw
- "valid polygon"
  - simple : edges cannot intersect
  - convex : no indentation
  - planar : vertices must lie on a plane
- all non-simple, non-convex, non-planar can be described by unions of valid polygons

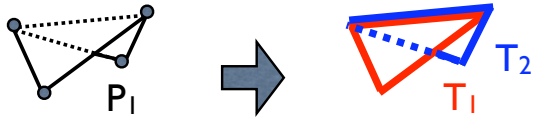
OpenGL Polygons

- Can be complicated and slow to draw
- "valid polygon"
  - simple : edges cannot intersect
  - convex : no indentation
  - planar : vertices must lie on a plane
- all non-simple, non-convex, non-planar can be described by unions of valid polygons



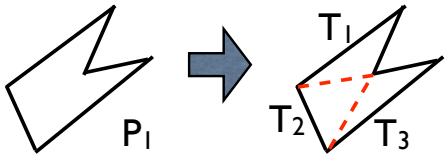
OpenGL Polygons

- Can be complicated and slow to draw
- "valid polygon"
  - simple : edges cannot intersect
  - convex : no indentation
  - planar : vertices must lie on a plane
- all non-simple, non-convex, non-planar can be described by unions of valid polygons



OpenGL Polygons

- Can be complicated and slow to draw
- "valid polygon"
  - simple : edges cannot intersect
  - convex : no indentation
  - planar : vertices must lie on a plane
- all non-simple, non-convex, non-planar can be described by unions of valid polygons



Coordinate Systems

- Object(Local) Coordinate System
  - orientation/alignment of object
  - useful in rotations
  - transformation of an object = opp. transformation of its coord system

Object coords don't change, world coordinates change

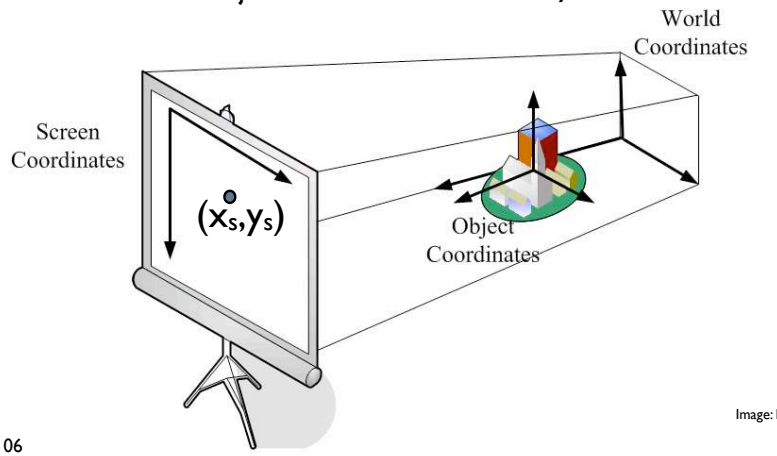


Image: Patrick Mauro

Coordinate Systems

- World Coordinate System
  - contains scene to be rendered
  - may be 2D(like a canvas) or 3D (like a box)

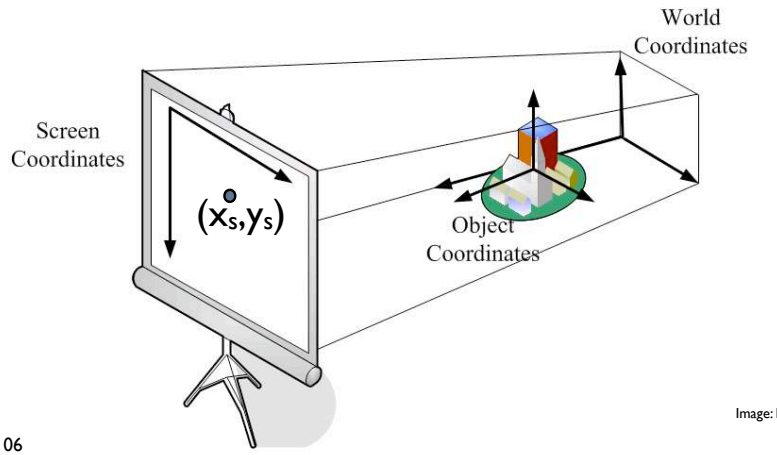
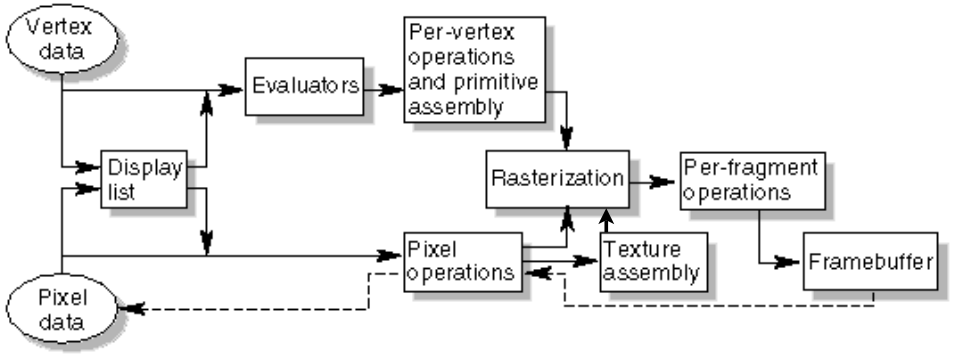


Image: Patrick Mauro

OpenGL Rendering Pipeline



Coordinate Systems

- Window(Screen) Coordinates
  - location on screen, counted in pixels
  - (0,0) upper left
  - range given by window size
  - mouse coords

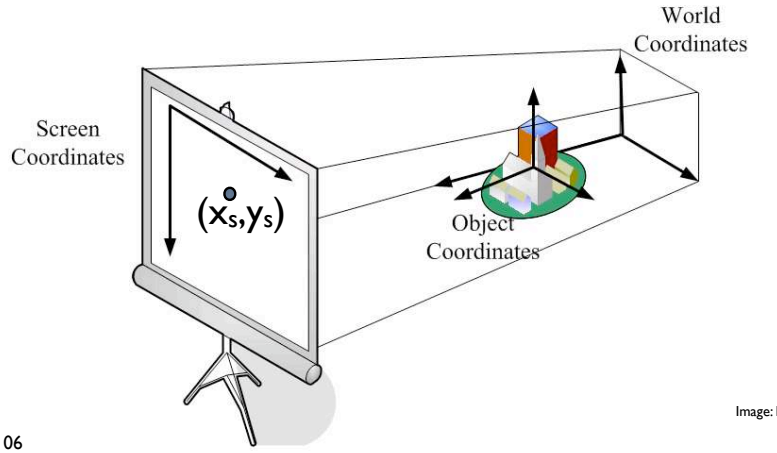
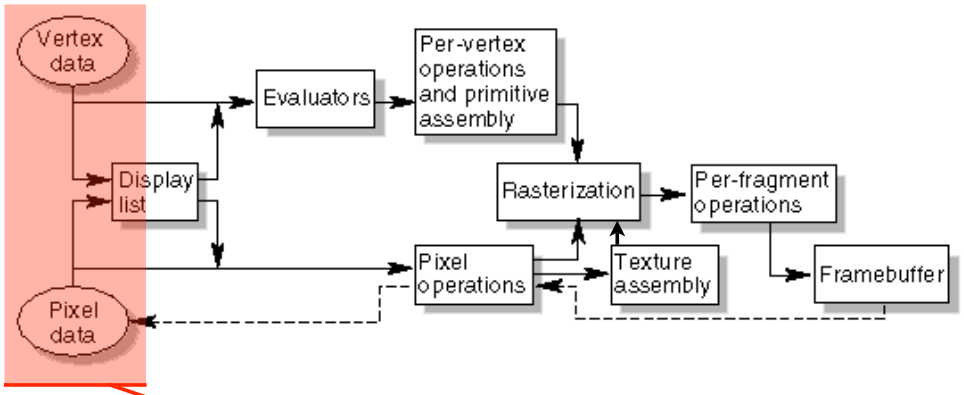
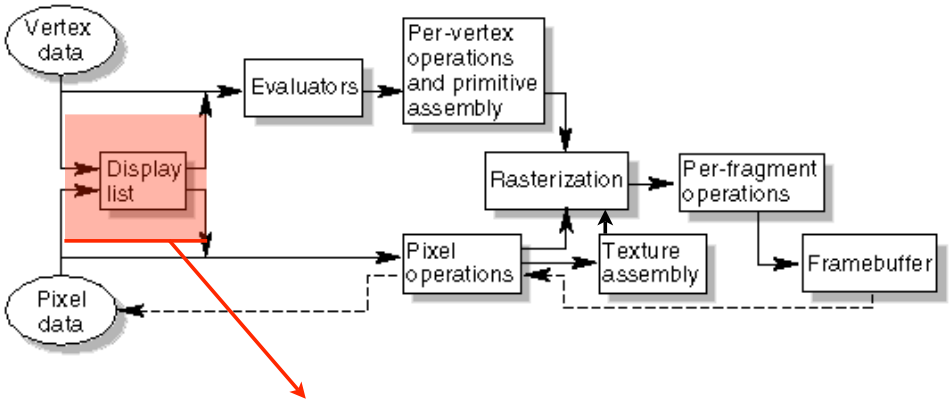


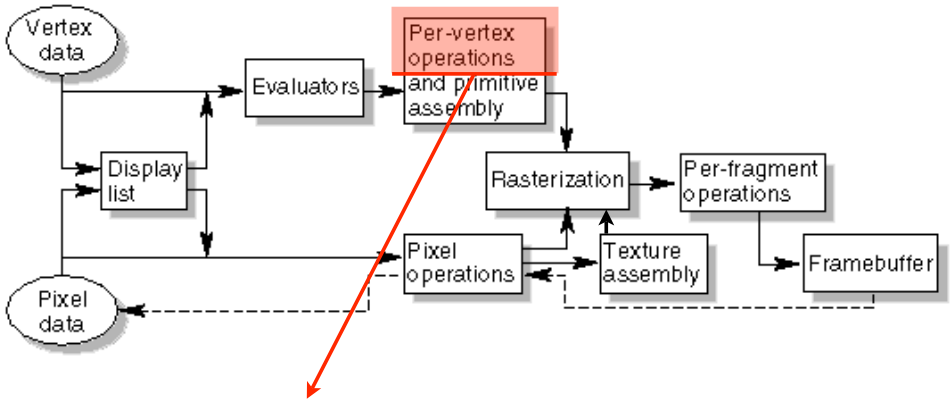
Image: Patrick Mauro



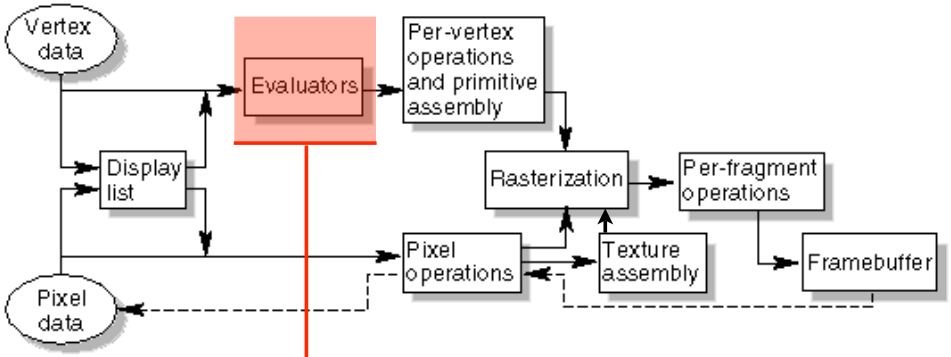
Initial Vertex Data : vertices, polygons, lines
Initial Pixel Data : pixels, bitmaps



general purpose data describing geometry/pixels
stores static geometry + attributes
for current(immediate) or later use
Speed-up

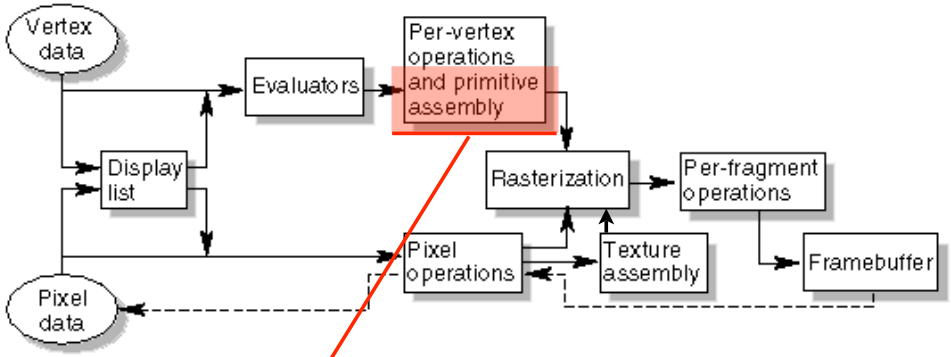


vertices to primitives
transformations
if texture : compute tex coords
if lighting : compute using position/normal/light source

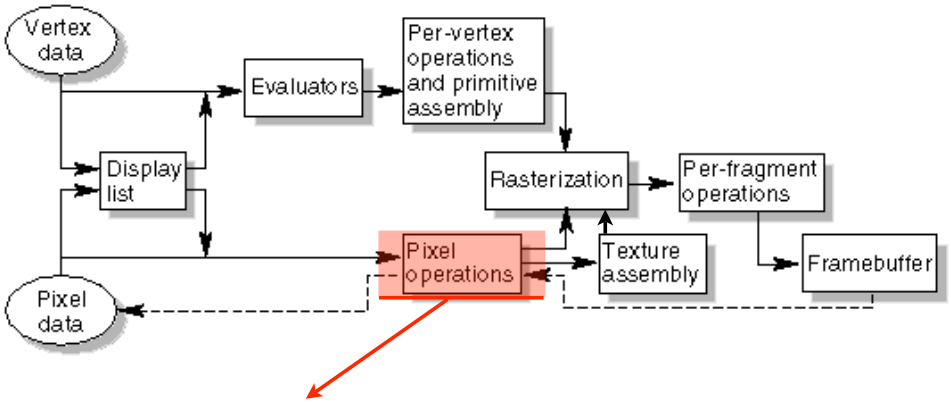


curves and surfaces in parametric form
control points: compact, easy to modify and store

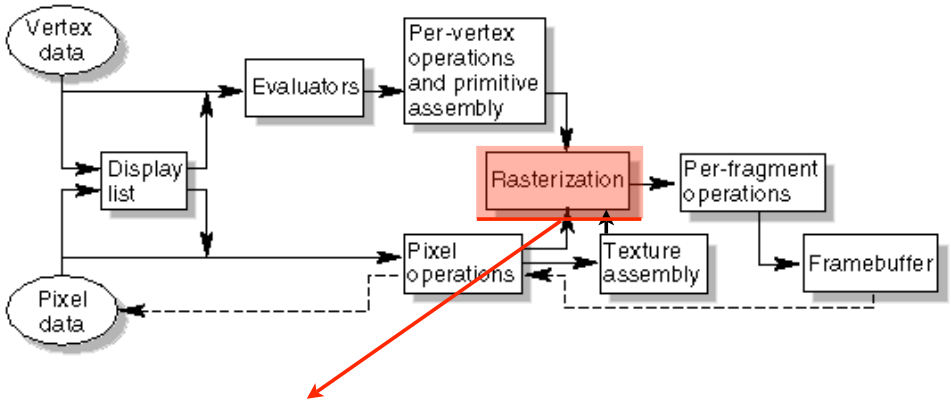
vertex data : spatial coords, normal, tex coords



clipping - application specific or view volume
  points : reject/accept
  line/polygon : new vertices
perspective division
viewport and depth operations
culling

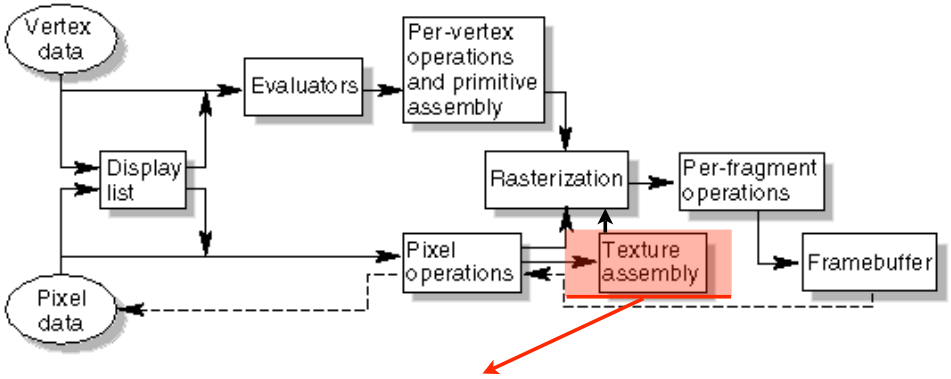


unpacking =>pixel map
or
pixel map => packing

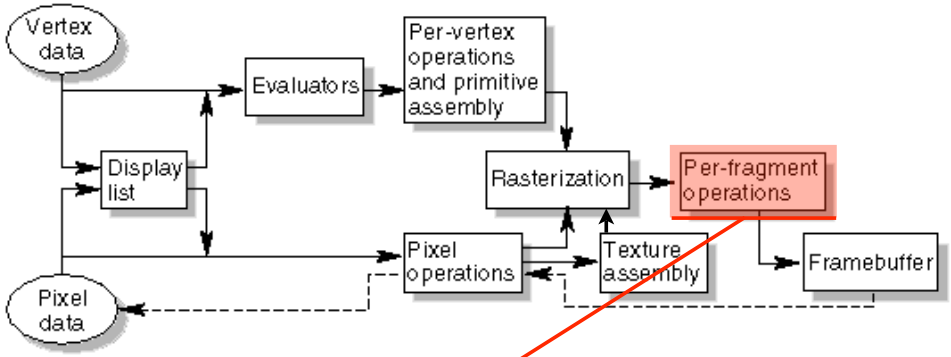


geometric/pixel data => fragments
each fragment is a pixel in the framebuffer

line width, points size, shading taken into account
antialiasing calculations
color, depth values



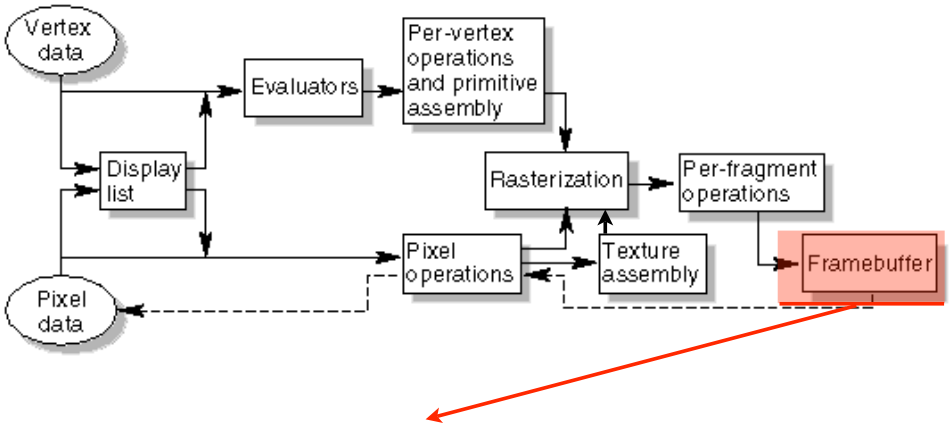
texture objects - for easy switch
"high performance texture memory" - prioritize!



modify/delete fragments before anything drawn

* tests such as depth buffer, fog, stencil, alpha
* blending/dithering operations

can all be enabled/disabled



finally draw onto frame buffer
to be rendered on screen

GLUT Basics

- Open GL Utilities Toolkit
- Basic window and interaction management

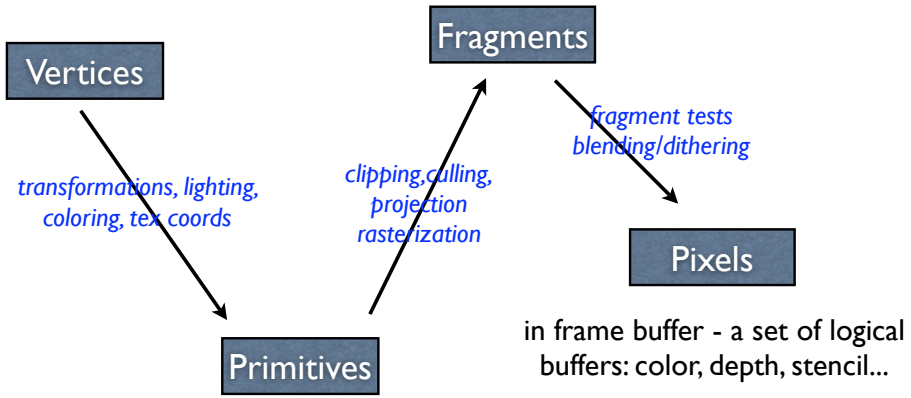
(GLUI an option for somewhat more complex user interface)

- glutInit(&argc, argv) - initialize GLUT
- glutInitDisplayMode(unsigned int mode)

GLUT_DOUBLE,
GLUT_DEPTH,
GLUT_RGB,
...

Summary

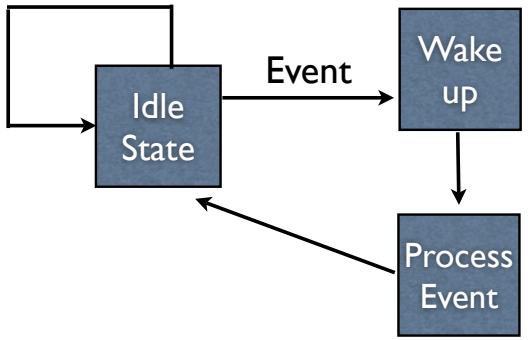
Each point in an image with
relevant info such as
color, depth, texture



in frame buffer - a set of logical
buffers: color, depth, stencil...

points, line segments, polygons
with relevant
edge flag, color and texture
info per vertex

Callbacks => event driven interaction (keyboard, mouse,...)



Why? - OpenGL operations generally
computationally expensive. Process only
as necessary

Keyboard events:

- glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));
- glutSpecialFunc(void (*func)(int key, int x, int y));

Mouse events:

- glutMouseFunc(void (*func)(int button, int state, int x, int y));
- glutMotionFunc(void (*func)(int x, int y));

Timer events:

- glutTimerFunc(uint msecs, void (*func)(int value), value));

Reshape events:

- glutReshapeFunc(void (*func)(int width, int height));

Idle events:

- glutIdleFunc(void (*func)(void));

- code examples
  - 2D & 3D Vector Classes
  - template.cpp
  - movingsquare.cpp

GLU Basics

- OpenGL Utility Library
- Can be used for:
  - Manipulation of images for use in texturing (image scale, automatic mipmapping)
  - Transformations (projection/viewing)
  - Polygon tessellation
  - Rendering quadrics (spheres,cones, etc.)
  - NURBS (sampling methods, trimming, etc.)