Computer Graphics

spring, 2013

Chapter 7
Primitive Assembly and Rasterization

Topics

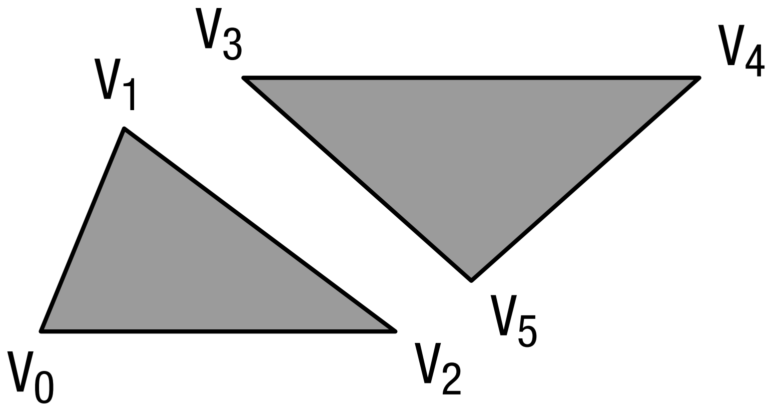
- Which primitives (geometric objects) are supported?
- How to draw them?
- What are done in the primitive assembly stage?
- What are done in the rasterization stage?

Primitives

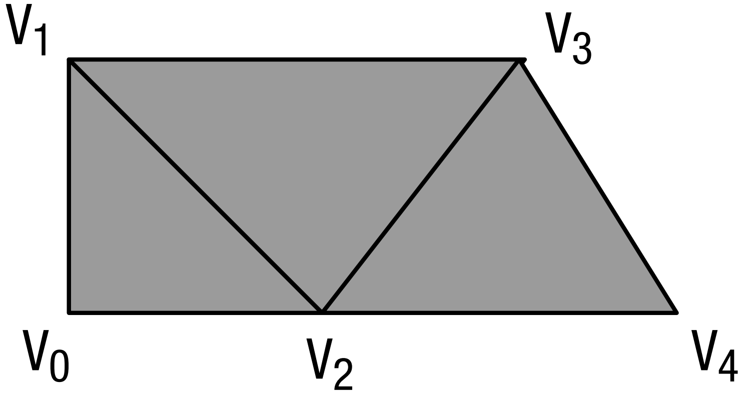
- Drawn using glDrawArrays / glDrawElements
- Described by a set of vertices (position, color, texcoords, normals, etc.)
- Types supported -- triangles, lines, points

Triangles

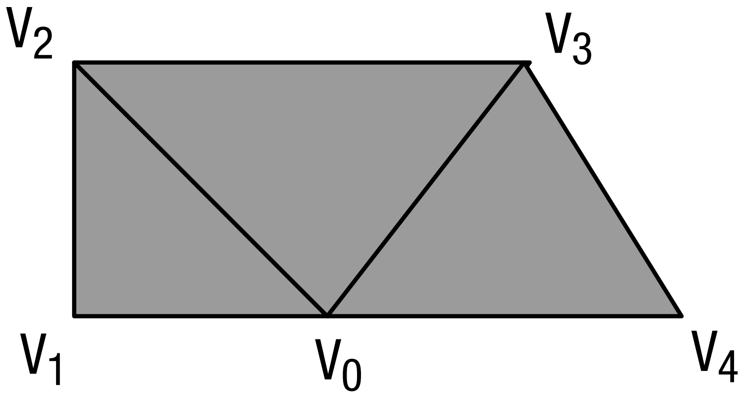
- Three type



GL_TRIANGLES



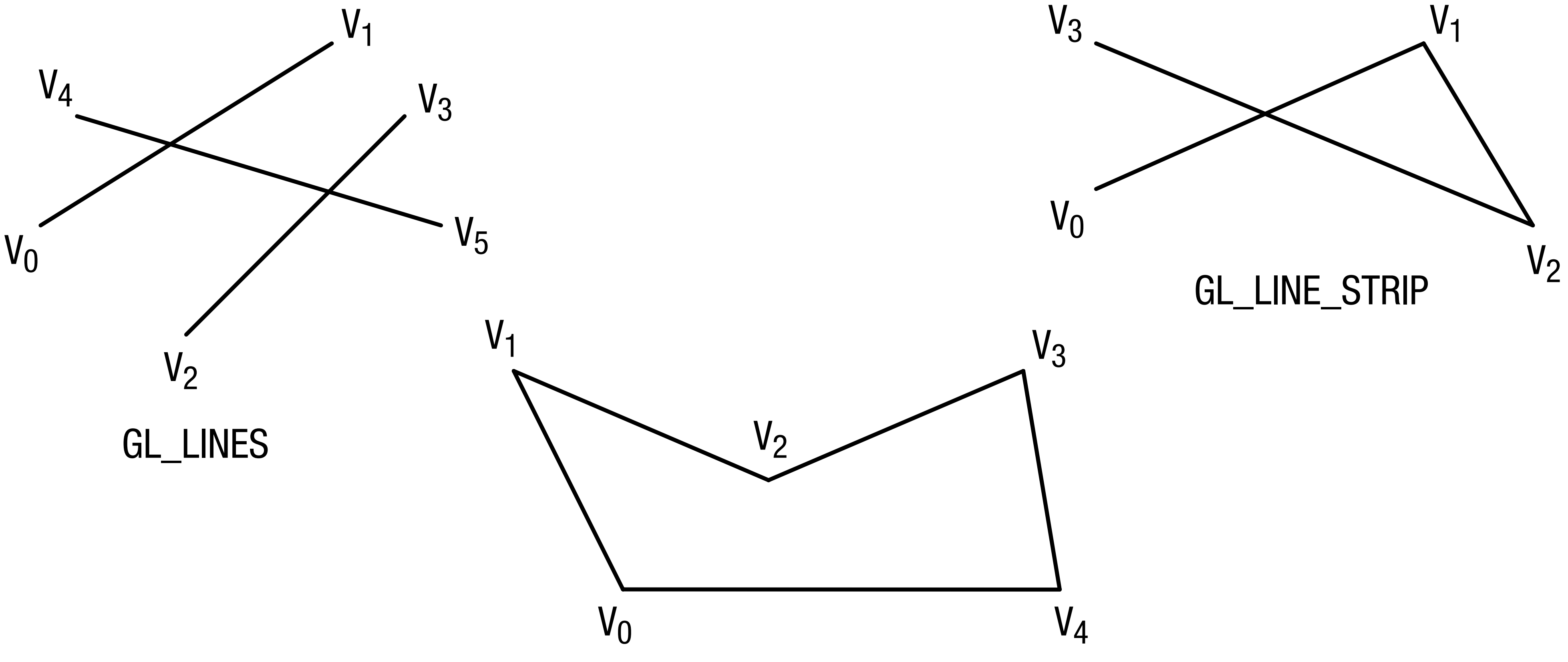
GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

Lines

- Three types
- Line width set by glLineWidth
- GL_ALIASED_LINE_WIDTH_RANGE



GL_LINES

GL_LINE_STRIP

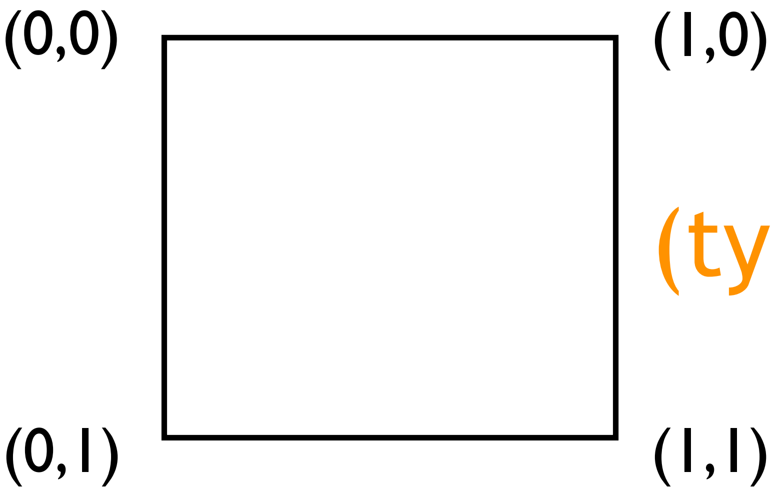
GL_LINE_LOOP

Point Stripes

- GL_POINTS
- Point stripe -- A screen-aligned quad specified by position & radius
- gl_PointSize -- built-in output variable in the vertex shader
- Point coord origin at (left,top) (cf: (left,bottom) for the window)

gl_PointCoord

- built-in read-only var in the fragment shader

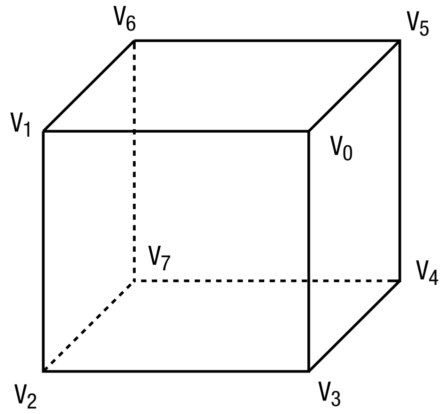


(typo in the textbook)

```
uniform sampler2D s_texSprite;

void
main(void)
{
   gl_FragColor = texture2D(s_texSprite, gl_PointCoord);
}
```

glDrawArrays

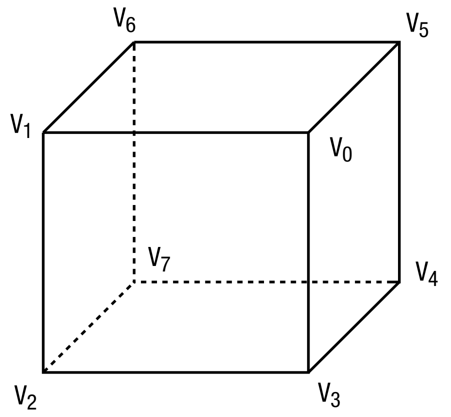
```
#define VERTEX_POS_INDX 0
#define NUM_FACES       6
GLfloat vertices[] = { … };  // (x, y, z) per vertex
glEnableVertexAttribArray(VERTEX_POS_INDX);
glVertexAttribPointer(VERTEX_POS_INDX, 3, GL_FLOAT, GL_FALSE,
                      0, vertices);
for (i=0; i<NUM_FACES; i++)
{
   glDrawArrays(GL_TRIANGLE_FAN, first, 4);
   first += 4;
}

          or

glDrawArrays(GL_TRIANGLES, 0, 36);
```



glDrawElements

```
#define VERTEX_POS_INDX 0
GLfloat vertices[] = { … };// (x, y, z) per vertex
GLubyte indices[36] = { 0, 1, 2, 0, 2, 3,
                        0, 3, 4, 0, 4, 5,
                        0, 5, 6, 0, 6, 1,
                        7, 1, 6, 7, 2, 1,
                        7, 5, 4, 7, 6, 5,
                        7, 3, 2, 7, 4, 3    };

glEnableVertexAttribArray(VERTEX_POS_INDX);
glVertexAttribPointer(VERTEX_POS_INDX, 3, GL_FLOAT, GL_FALSE,
                      0, vertices);
glDrawElements(GL_TRIANGLES, sizeof(indices)/sizeof(GLubyte),
               GL_UNSIGNED_BYTE, indices);
```

(typo in the textbook)

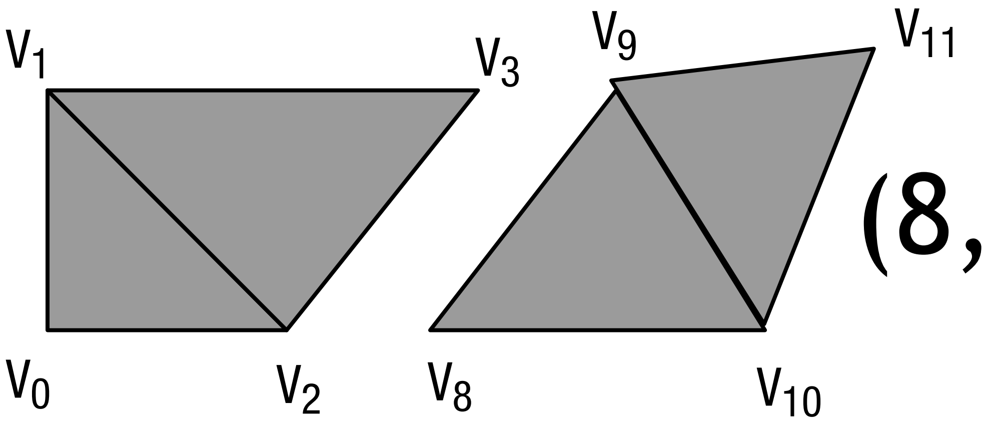


Performance Tips

- glDrawElements for GL_TRIANGLES
- Multiple tri strips or fans can be merged into one using degenerate triangles --> degenerate ones are detected & rejected by GPU
- The way adding vertices needs care due to winding orders

Opposite Vertex Order

(0,1,2,3)



(8,9,10,11)

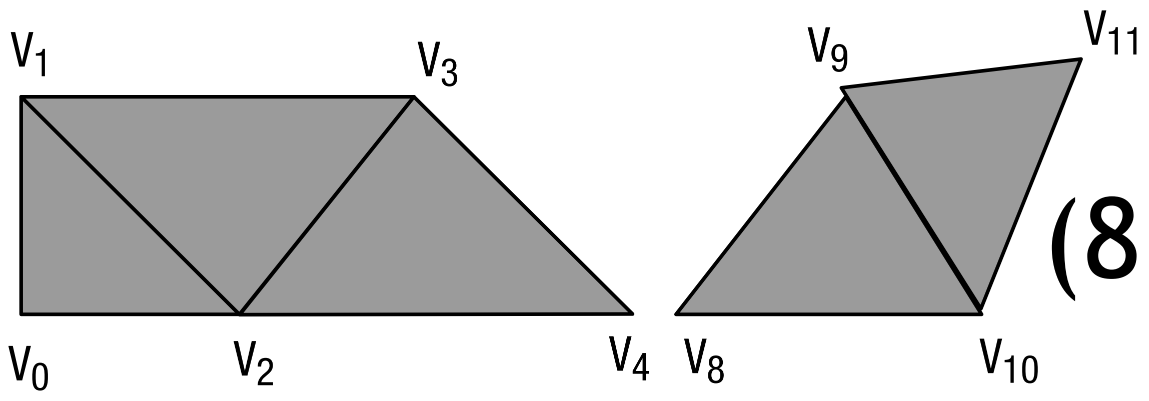
(0,1,2),(1,2,3),(8,9,10),(9,10,11)

(0,1,2,3,3,8,8,9,10,11)

(0,1,2),(1,2,3),(3,3,8),(8,8,9),(8,9,10),(9,10,11)

Same Vertex Order

(0,1,2,3,4)



(8,9,10,11)

(0,1,2),(1,2,3),(2,3,4),(8,9,10),(9,10,11)

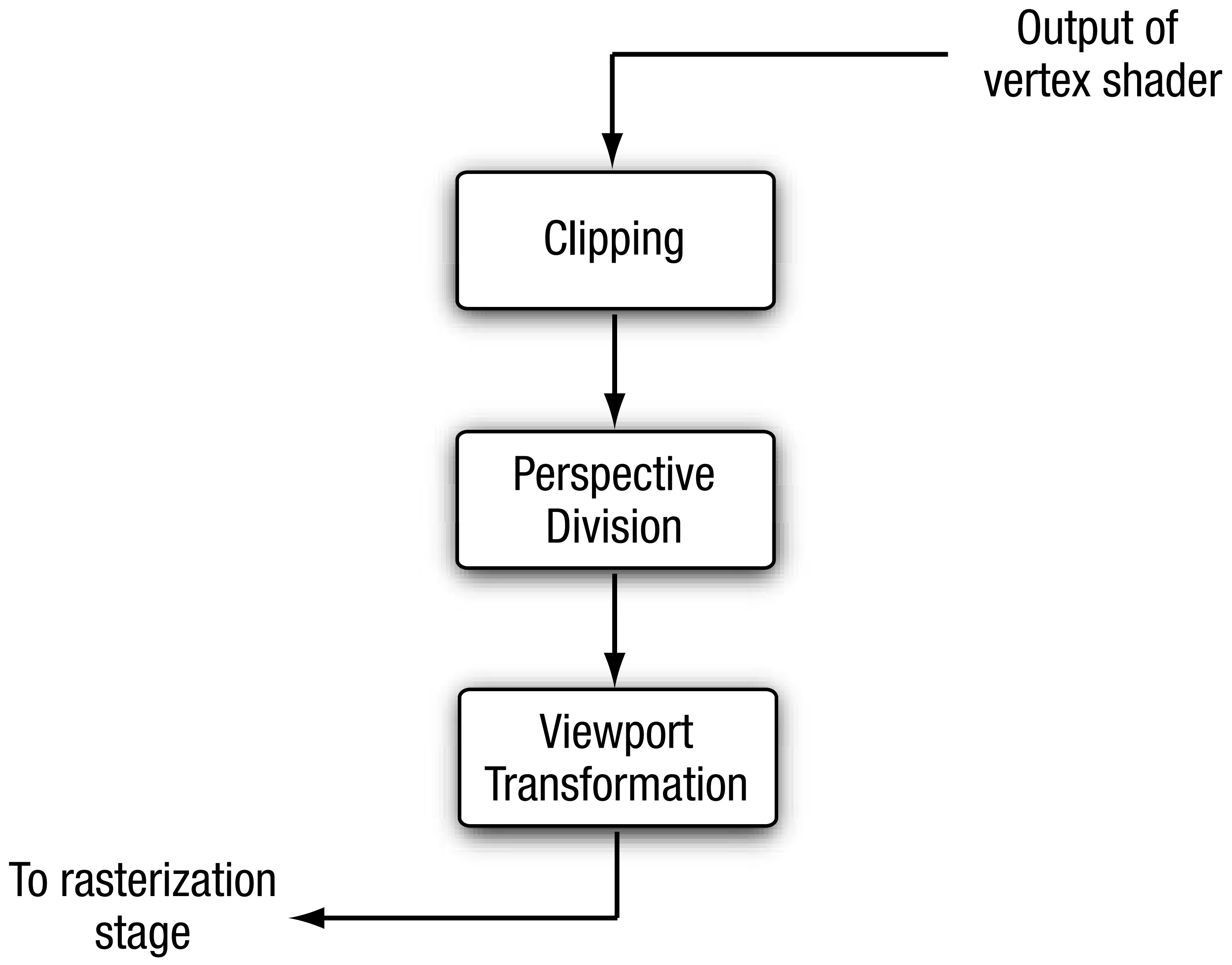
(typo in the textbook)

(0,1,2,3,4,4,4,8,8,9,10,11)
(0,1,2),(1,2,3),(2,3,4),(3,4,4),(4,4,4),(4,4,8),
(8,8,9),(8,9,10),(9,10,11)

Post-Transform Vertex Cache

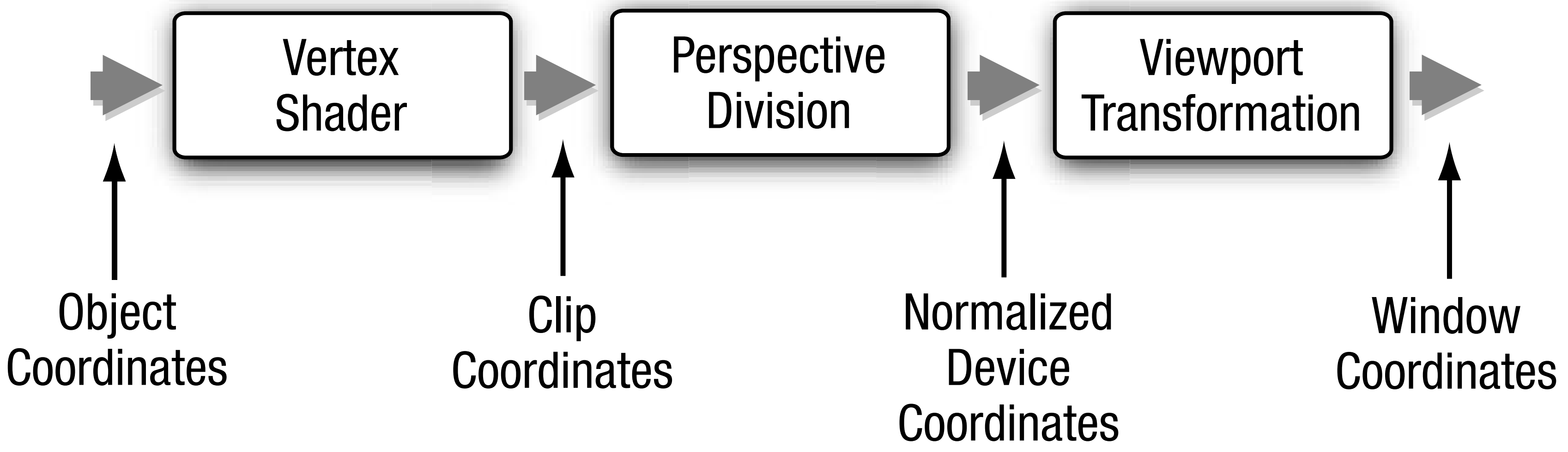
- Memory area where processed vertices (but before primitive) are cached
- If the same vertex exists in the cache, it is not processed --> tested by the index --> indexed rendering required
- More at https://www.opengl.org/wiki/Post_Transform_Cache

Primitive Assembly



Coordinate Systems

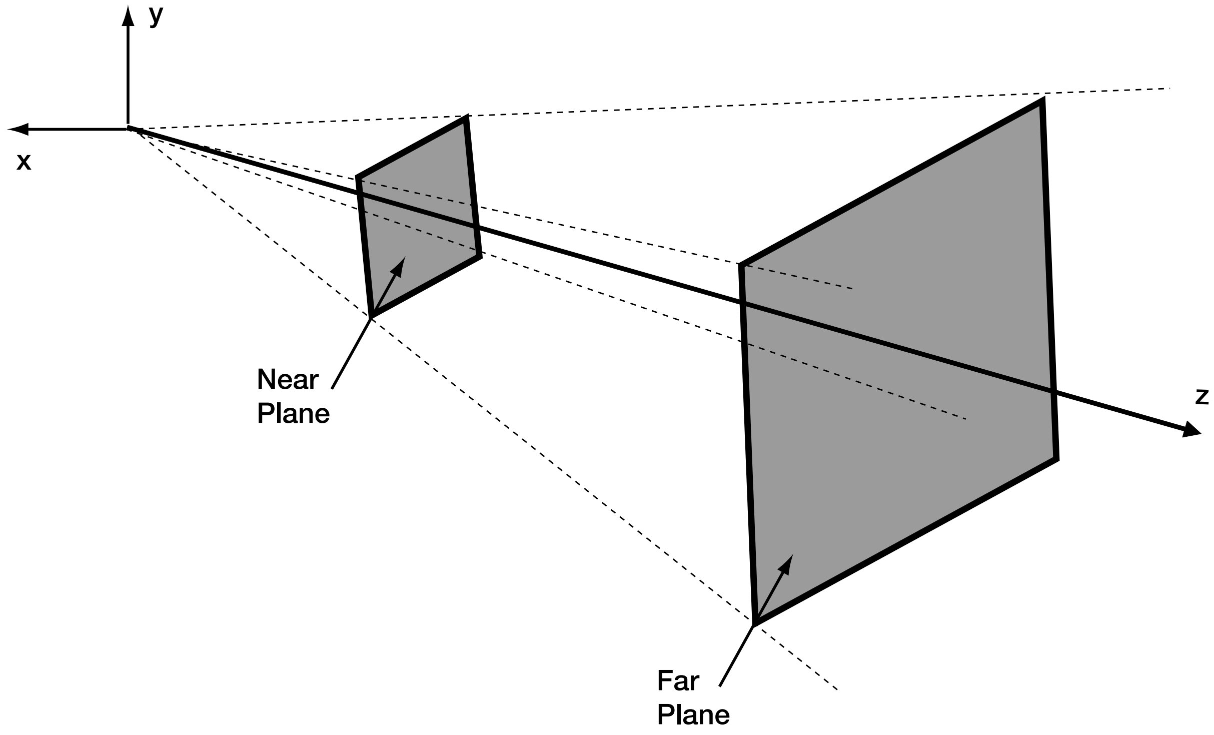
- Obj coords --> clip coords: done by loading & multiplying appropriate matrices in the vertex shader (Ch 8)
- Other transformations are done by GPU



Clipping

- To clip vertices outside view volume (a.k.a. clip volume)
- Done in clip coords (xc,yc,zc,wc)
- The clip volume is defined as

$$-w_c <= x_c <= w_c$$
$$-w_c <= y_c <= w_c$$
$$-w_c <= z_c <= w_c$$



Clipping (cont’d)

- Clipping triangles -- new vertices may be generated and the triangle is converted to a triangle fan
- Clipping lines -- new vertices may be generated
- Clipping points -- may be scissored

Clipping (cont’d)

- Clipping may be expensive
- Clipping against x & y planes may be done by scissoring test (which is implemented very efficiently by GPUs) & the viewport with the guard-band region

Perspective Division

- (xc,yc,zc,wc) in Clip coords

  --> (xc/wc,yc/wc,zc/wc) in NDC (Normalized Device Coordinates)
- NDC defined in [-1,1]x[-1,1]x[-1,1]
- (xd,yd)=(xc/wc,yc/wc) is converted to window coords by viewport transformation
- zd=zc/wc is converted to screen z value (depth) using near & far values set by glDepthRange

Viewport Transformation

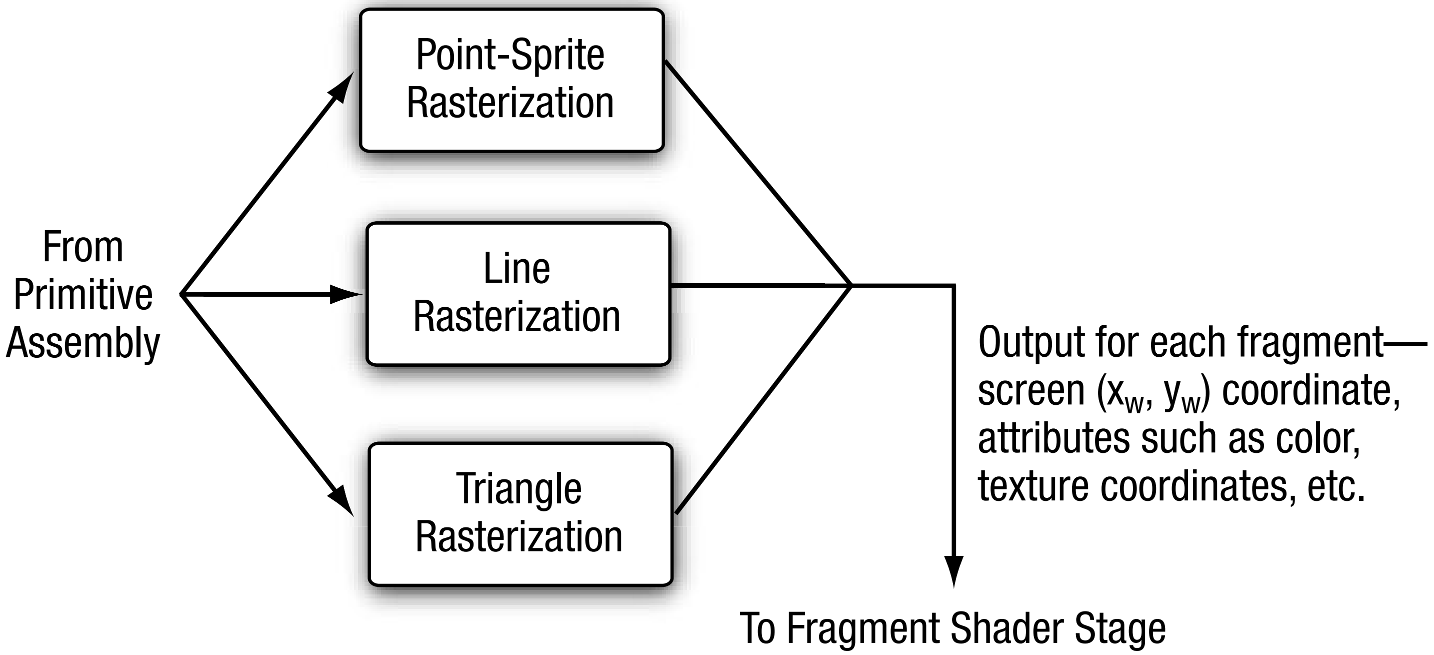
- Set by glViewport -- default is the window size
- Transformation

$$\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} (w/2)x_d + o_x \\ (h/2)y_d + o_y \\ ((f-n)/2)z_d + (n+f)/2 \end{bmatrix}$$

  - ox=x+w/2, oy=y+h/2 (typo in the textbook)
  - n & f are set by glDepthRangef

Rasterization

- Primitives are rasterized into fragments (Ch 9 & 10)
- Varying variables are linearly interpolated

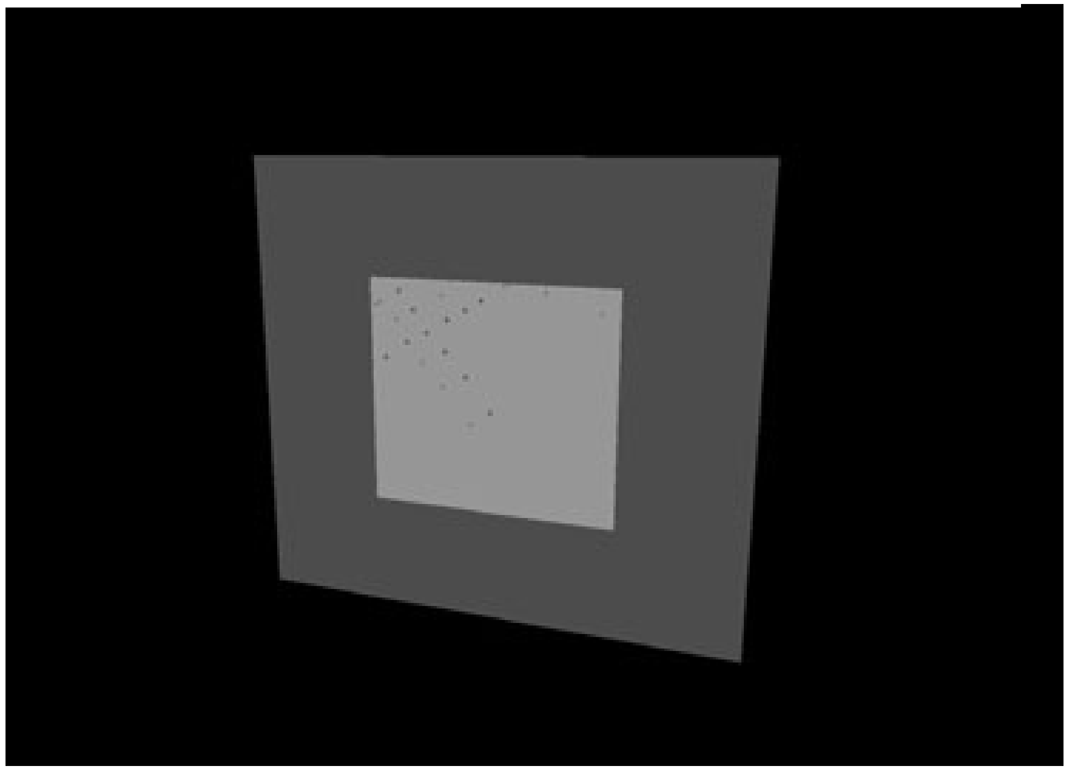

From Primitive Assembly
Point-Sprite Rasterization
Line Rasterization
Triangle Rasterization
Output for each fragment— screen (x_w, y_w) coordinate, attributes such as color, texture coordinates, etc.
To Fragment Shader Stage

Culling

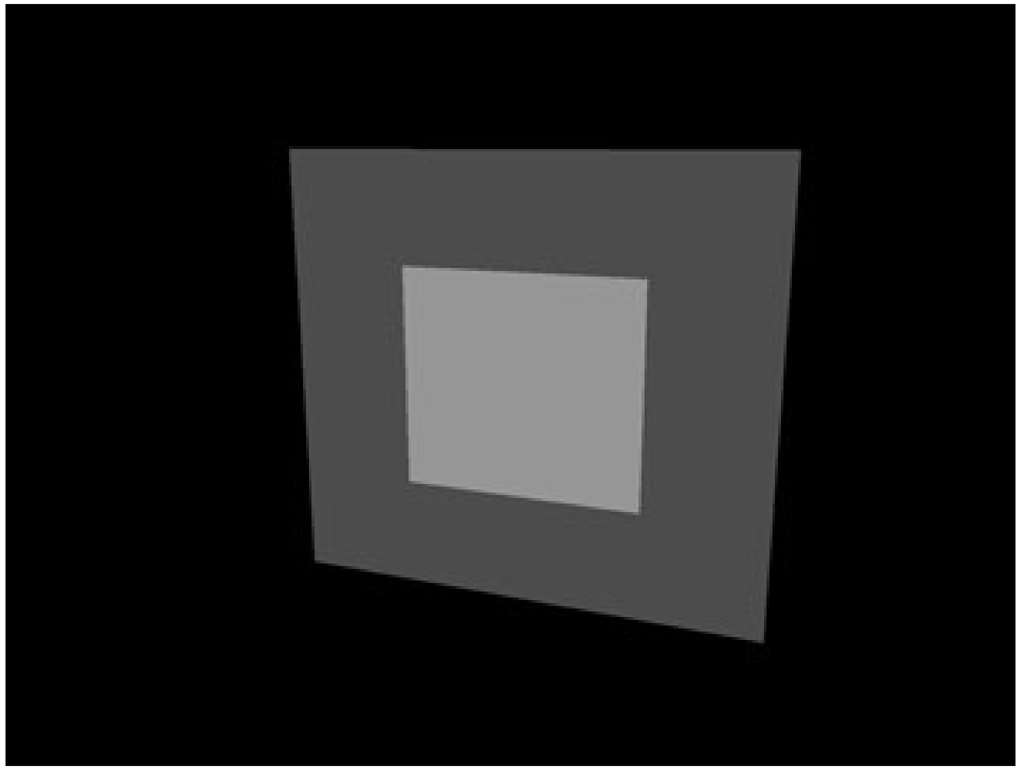
- Triangles are discarded depending on their facing direction -- glCullFace
- Orientation (winding order) set by glFrontFace -- CW or CCW
- Enabled/disabled with GL_CULL_FACE --> Always enable for better performance!

Polygon Offset

- To avoid z-fighting artifacts
- Original depth + delta used for depth test
- Original depth value is stored



Polygon Offset Disabled



Polygon Offset Enabled

Polygon Offset

- Set by glPolygonOffset, enabled by GL_POLYGON_OFFSET_FILL
- depth offset = m * *factor* + r * *units*
  - $m = \sqrt{(\partial z/\partial x^2 + \partial z/\partial y^2)}$ or $\max\{|\partial z/\partial x|, |\partial z/\partial y|\}$
  - r: smallest value that can produce a guaranteed difference in depth value (implementation-dependent)

Polygon Offset (cont’d)

```
const float polygonOffsetFactor = -1.0f;
const float polygonOffsetUnits  = -2.0f;

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// load vertex shader
// set the appropriate transformation matrices
// set the vertex attribute state

// draw the RED triangle
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);

// set the depth func to <= as polygons are coplanar
glDepthFunc(GL_LEQUAL);

glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(polygonOffsetFactor, polygonOffsetUnits);
// set the vertex attribute state

// draw the GREEN triangle
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```