



# OpenGL suite et Mini Projet

Déjà présentés : introduction générale et OpenGL (éléments de base, géométrie, transformations)





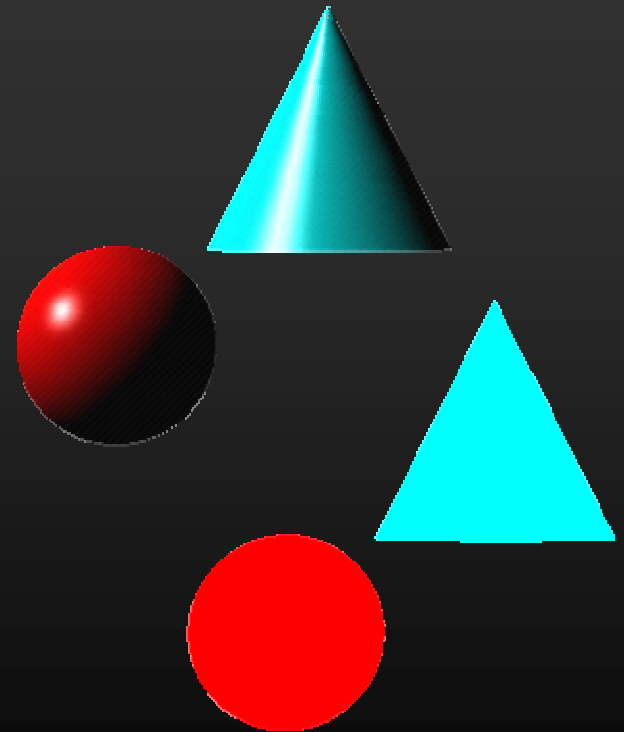
# Lighting

# Lighting Principles



- **Lighting simulates how objects reflect light**

- material composition of object
- light's color and position
- global lighting parameters
  - ambient light
  - two sided lighting
- available in both color index and RGBA mode

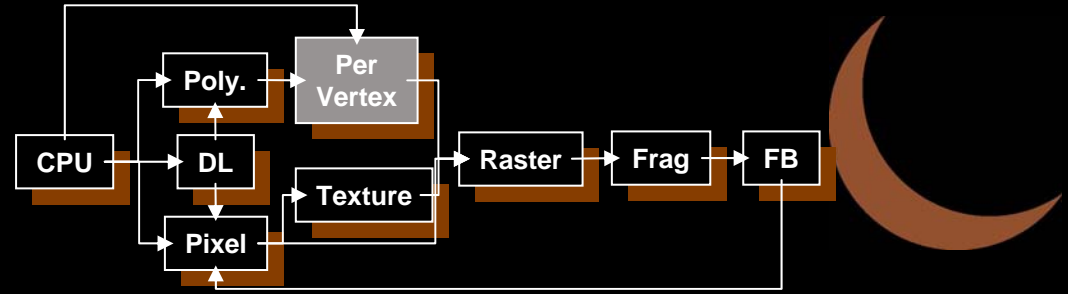


# How OpenGL Simulates Lights



- **Phong lighting model**
  - Computed at vertices
  - For per-pixel computations, needs new hardware (next session)
- **Lighting contributors**
  - Surface material properties
  - Light properties
  - Lighting model properties

# Surface Normals



- Normals define how a surface reflects light

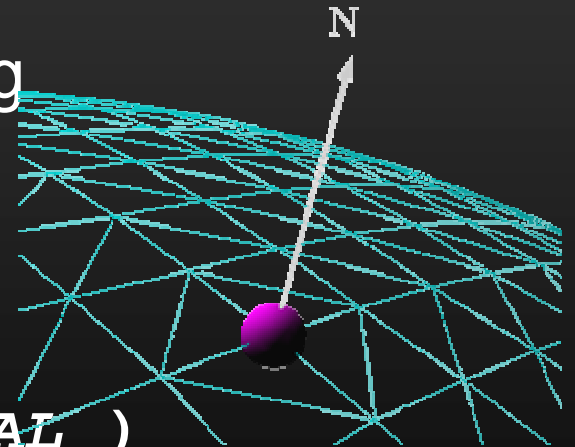
`glNormal3f( x, y, z )`

- Current normal is used to compute vertex's color
- Use *unit* normals for proper lighting
  - scaling affects a normal's length

`glEnable( GL_NORMALIZE )`

or

`glEnable( GL_RESCALE_NORMAL )`



# Material Properties



- Define the surface properties of a primitive  
`glMaterialfv( face, property, value );`

<code>GL_DIFFUSE</code>	Base color
<code>GL_SPECULAR</code>	Highlight Color
<code>GL_AMBIENT</code>	Low-light Color
<code>GL_EMISSION</code>	Glow Color
<code>GL_SHININESS</code>	Surface Smoothness

- separate materials for front and back
  - Face can be `GL_FRONT` or `GL_BACK`

# Light Properties



```
glLightfv( light, property, value );
```

- *light* specifies which light
  - multiple lights, starting with `GL_LIGHT0`

```
glGetIntegerv( GL_MAX_LIGHTS, &n );
```

- *properties*
  - colors
  - position and type
  - attenuation

# Light Sources (cont.)



- **Light color properties**

- **GL\_AMBIENT**
- **GL\_DIFFUSE**
- **GL\_SPECULAR**



# Types of Lights



- **OpenGL supports two types of Lights**
  - Local (Point) light sources
  - Infinite (Directional) light sources
- **Type of light controlled by  $w$  coordinate**
  - $w = 0$  *Infinite Light directed along  $(x \ y \ z)$*
  - $w \neq 0$  *Local Light positioned at  $(\frac{x}{w} \ \frac{y}{w} \ \frac{z}{w})$*

# Turning on the Lights



- Flip each light's switch

```
glEnable( GL_LIGHTn );
```

- Turn on the power

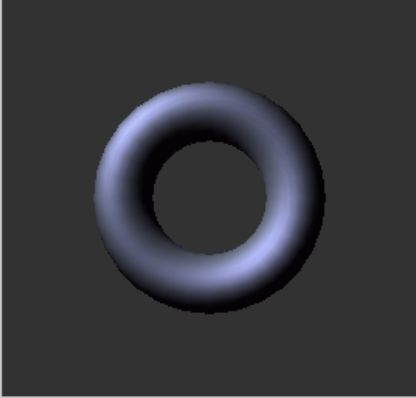
```
glEnable( GL_LIGHTING );
```

# Light Material Tutorial

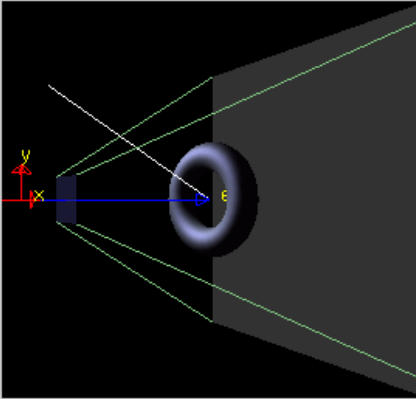


Light & Material

Screen-space view



World-space view



Command manipulation window

```
GLfloat light_pos[ ] = { -2.00 , 2.00 , 2.00 , 1.00 };
GLfloat light_Ka[ ] = { 0.00 , 0.00 , 0.00 , 1.00 };
GLfloat light_Kd[ ] = { 1.00 , 1.00 , 1.00 , 1.00 };
GLfloat light_Ks[ ] = { 1.00 , 1.00 , 1.00 , 1.00 };

glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_Ka);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_Kd);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_Ks);

GLfloat material_Ka[ ] = { 0.11 , 0.06 , 0.11 , 1.00 };
GLfloat material_Kd[ ] = { 0.43 , 0.47 , 0.54 , 1.00 };
GLfloat material_Ks[ ] = { 0.33 , 0.33 , 0.52 , 1.00 };
GLfloat material_Ke[ ] = { 0.00 , 0.00 , 0.00 , 0.00 };
GLfloat material_Se = 10 ;

glMaterialfv(GL_FRONT, GL_AMBIENT, material_Ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, material_Kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, material_Ks);
glMaterialfv(GL_FRONT, GL_EMISSION, material_Ke);
glMaterialfv(GL_FRONT, GL_SHININESS, material_Se);
```

Click on the arguments and move the mouse to modify values.

# Controlling a Light's Position



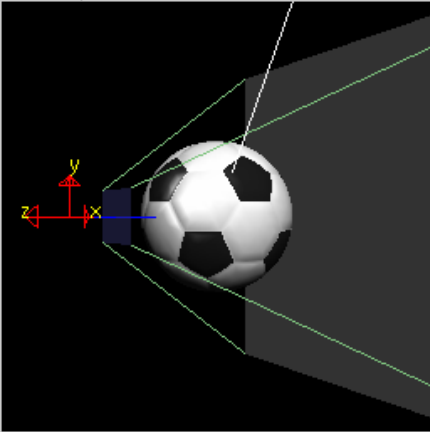
- **Modelview matrix affects a light's position**
  - Different effects based on when position is specified
    - eye coordinates
    - world coordinates
    - model coordinates
  - Push and pop matrices to uniquely control a light's position

# Light Position Tutorial




Light Positioning

World-space view



Screen-space view



Command manipulation window

```
GLfloat pos[4] = { 1.50 , 1.00 , 1.00 , 0.00 };  
gluLookAt( 0.00 , 0.00 , 2.00 , <- eye  
          0.00 , 0.00 , 0.00 , <- center  
          0.00 , 1.00 , 0.00 ); <- up  
glLightfv(GL_LIGHT0, GL_POSITION, pos);
```

Click on the arguments and move the mouse to modify values.

# Advanced Lighting Features



- **Spotlights**
  - localize lighting affects
    - *GL\_SPOT\_DIRECTION*
    - *GL\_SPOT\_CUTOFF*
    - *GL\_SPOT\_EXPONENT*

# Advanced Lighting Features



- **Light attenuation**

- decrease light intensity with distance
  - *GL\_CONSTANT\_ATTENUATION*
  - *GL\_LINEAR\_ATTENUATION*
  - *GL\_QUADRATIC\_ATTENUATION*

$$f_i = \frac{1}{k_c + k_l d + k_q d^2}$$

# Light Model Properties



```
glLightModelfv( property, value );
```

- Enabling two sided lighting

```
GL_LIGHT_MODEL_TWO_SIDE
```

- Global ambient color

```
GL_LIGHT_MODEL_AMBIENT
```

- Local viewer mode

```
GL_LIGHT_MODEL_LOCAL_VIEWER
```

- Separate specular color

```
GL_LIGHT_MODEL_COLOR_CONTROL
```



# Tips for Better Lighting



- **Recall lighting computed only at vertices**
  - model tessellation heavily affects lighting results
    - better results but more geometry to process
- **Use a single infinite light for fastest lighting**
  - minimal computation per vertex



# Imaging and Raster Primitives



# Imaging and Raster Primitives



- Describe OpenGL's raster primitives: bitmaps and image rectangles
- Demonstrate how to get OpenGL to read and render pixel rectangles

# Pixel-based primitives



- **Bitmaps**

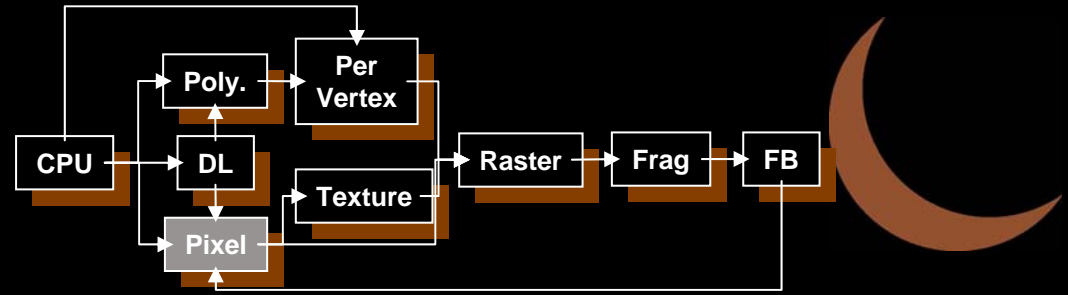
- 2D array of bit masks for pixels
  - update pixel color based on current color

- **Images**

- 2D array of pixel color information
  - complete color information for each pixel

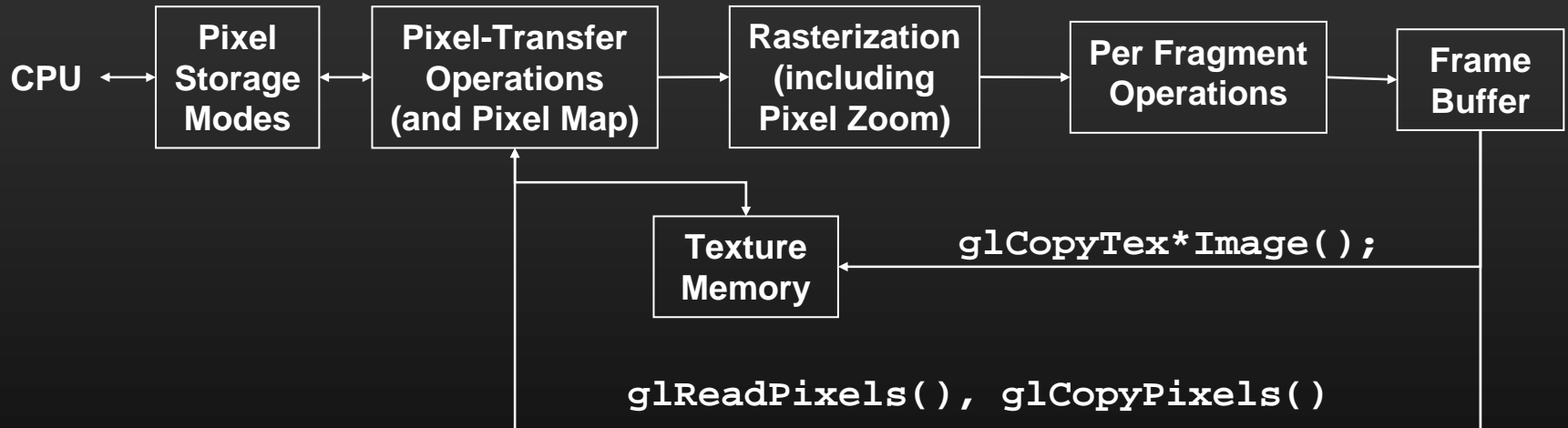
- **OpenGL doesn't understand image formats**

# Pixel Pipeline



- **Programmable pixel storage and transfer operations**

`glBitmap()`, `glDrawPixels()`



# Rendering Images



```
glDrawPixels( width, height, format, type,  
             pixels )
```

- render pixels with lower left of image at current raster position
- numerous formats and data types for specifying storage in memory
  - best performance by using format and type that matches hardware



# Reading Pixels



```
glReadPixels( x, y, width, height, format,  
              type, pixels )
```

- read pixels from specified (x,y) position in framebuffer
- pixels automatically converted from framebuffer format into requested format and type
- **Framebuffer pixel copy**

```
glCopyPixels( x, y, width, height, type )
```

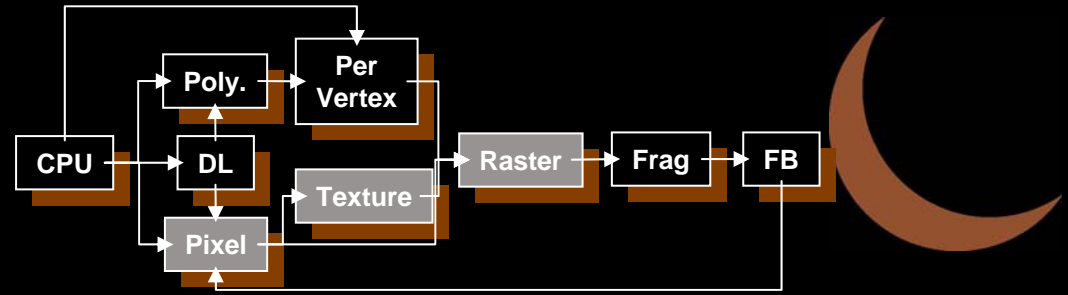


# Texture Mapping



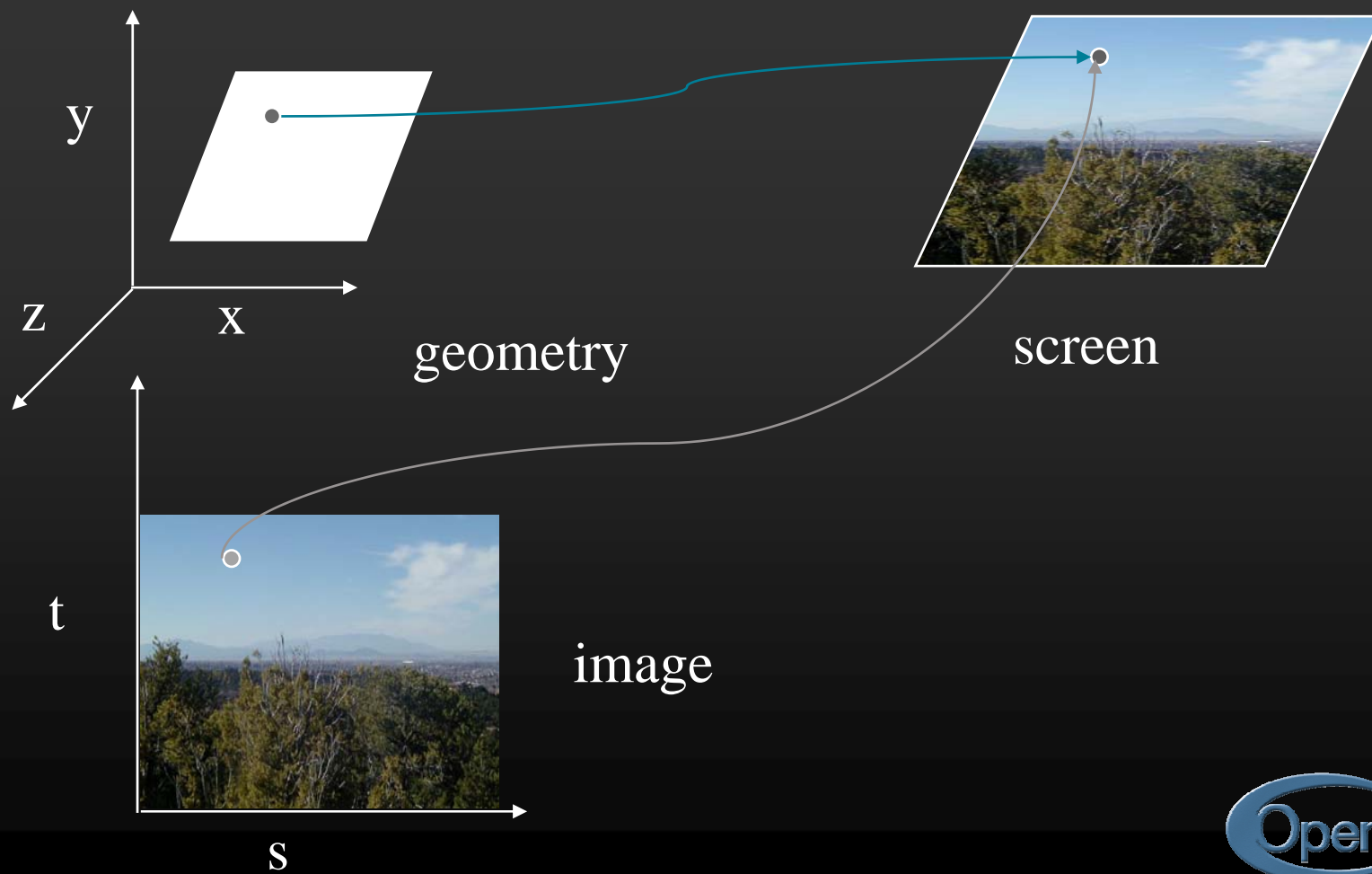


# Texture Mapping



- **Apply a 1D, 2D, or 3D image to geometric primitives**
- **Uses of Texturing**
  - simulating materials
  - reducing geometric complexity
  - image warping
  - reflections

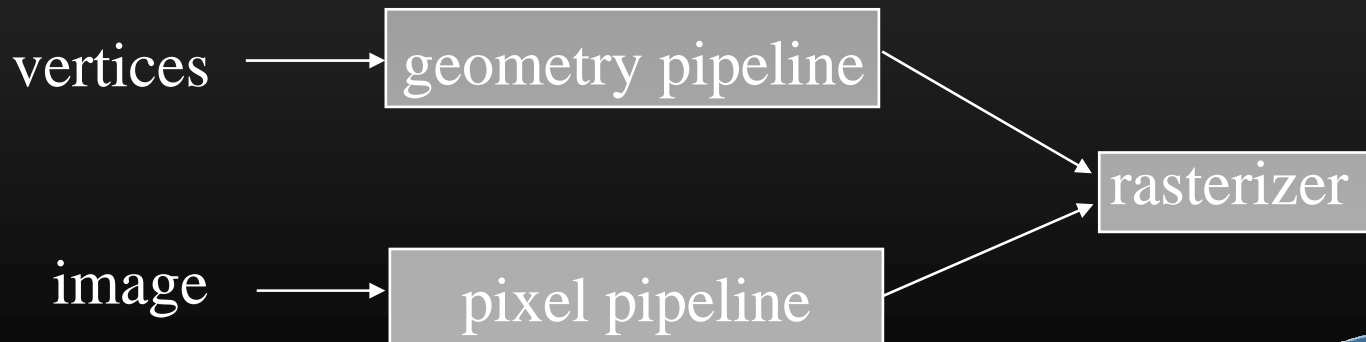
# Texture Mapping



# Texture Mapping and the OpenGL Pipeline



- Images and geometry flow through separate pipelines that join at the rasterizer
  - “complex” textures do not affect geometric complexity



# Texture Example



- The texture (below) is a 256 x 256 image that has been mapped to a rectangular polygon which is viewed in perspective



# Applying Textures I



- **Three steps**

- ① specify texture

- read or generate image
- assign to texture

- ② assign texture coordinates to vertices

- ③ specify texture parameters

- wrapping, filtering

# Applying Textures II



- specify textures in texture objects
- set texture filter
- set texture function
- set texture wrap mode
- set optional perspective correction hint
- bind texture object
- enable texturing
- supply texture coordinates for vertex
  - coordinates can also be generated

# Texture Objects



- **Like display lists for texture images**
  - one image per texture object
  - may be shared by several graphics contexts

- **Generate texture names**

```
glGenTextures( n, *texIds );
```

# Texture Objects (cont.)



- Create texture objects with texture data and state

```
glTexImage2
```

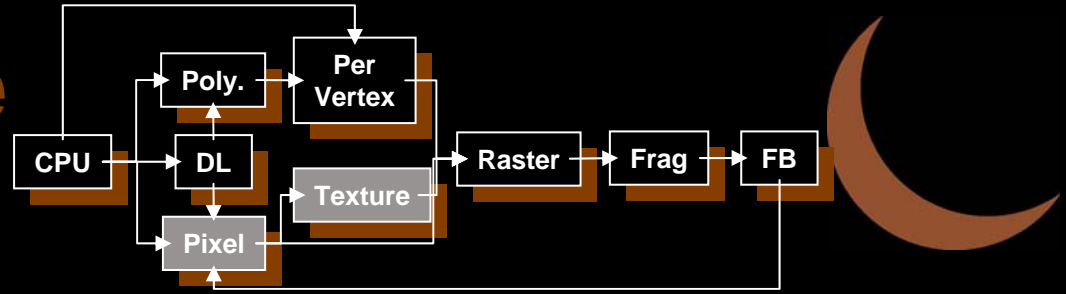
```
glTextureParameteri
```

- Bind textures before using

```
glBindTexture( target, id );
```



# Specify Texture Image



- Define a texture image from an array of texels in CPU memory

```
glTexImage2D( target, level, components,  
             w, h, border, format, type, *texels );
```

- dimensions of image must be powers of 2
- **Texel colors are processed by pixel pipeline**
  - pixel scales, biases and lookups can be done

# Converting A Texture Image



- If dimensions of image are not power of 2  
`gluScaleImage( format, w_in, h_in,  
                  type_in, *data_in, w_out, h_out,  
                  type_out, *data_out );`
  - \*\_in *is for source image*
  - \*\_out *is for destination image*
- Image interpolated and filtered during scaling

# Specifying a Texture: Other Methods



- **Use frame buffer as source of texture image**
  - uses current buffer as source image

`glCopyTexImage2D(...)`

`glCopyTexImage1D(...)`

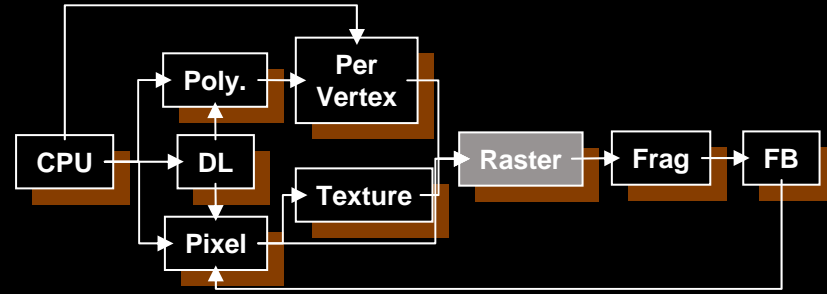
- **Modify part of a defined texture**

`glTexSubImage2D(...)`

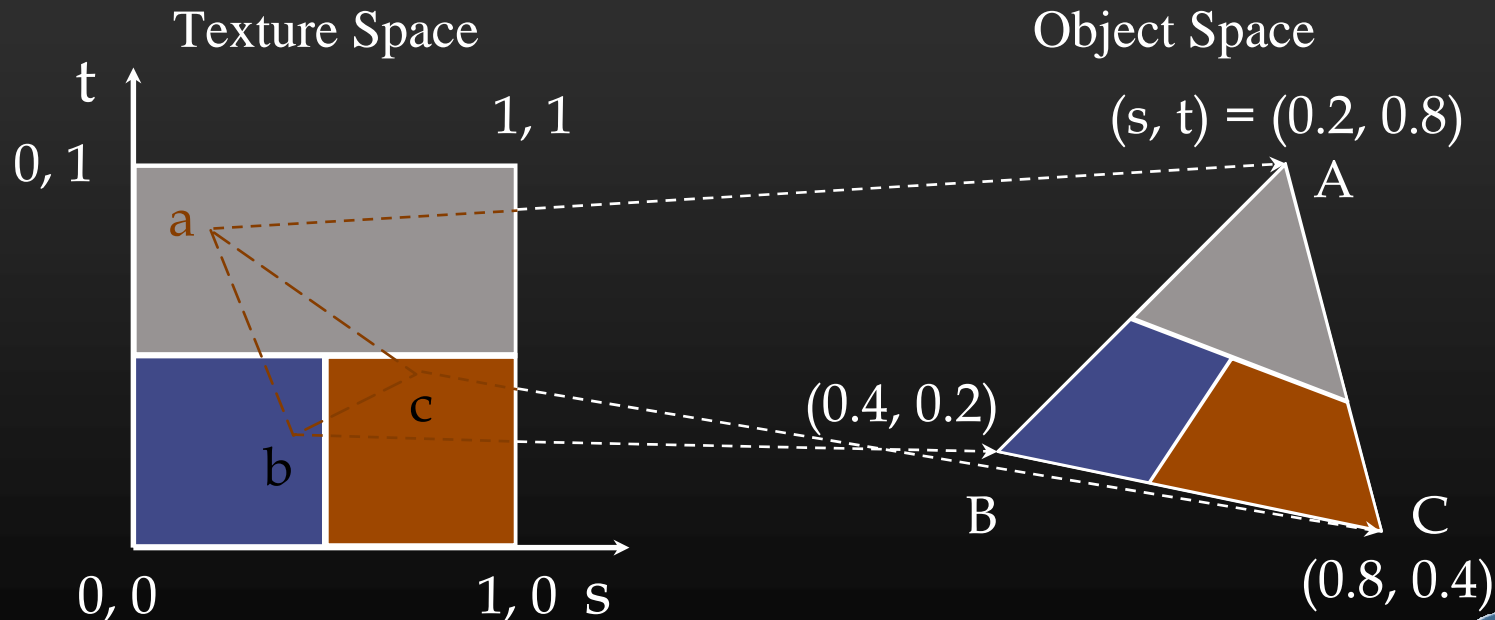
`glTexSubImage1D(...)`

- **Do both with `glCopyTexSubImage2D(...)`, etc.**

# Mapping a Texture



- Based on parametric texture coordinates
- `glTexCoord*()` specified at each vertex



# Generating Texture Coordinates



- **Automatically generate texture coords**

```
glTexGen{ifd}[v]()
```

- **specify a plane**

- generate texture coordinates based upon distance from plane  $Ax + By + Cz + D = 0$

- **generation modes**

- **GL\_OBJECT\_LINEAR**
- **GL\_EYE\_LINEAR**
- **GL\_SPHERE\_MAP**

# Texture Application Methods



- **Filter Modes**

- minification or magnification
- special mipmap minification filters

- **Wrap Modes**

- clamping or repeating

- **Texture Functions**

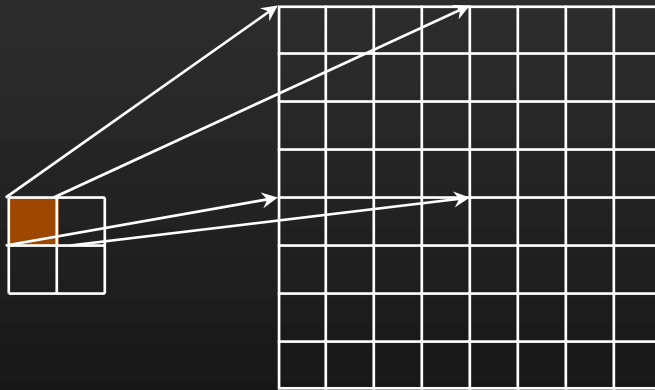
- how to mix primitive's color with texture's color
  - blend, modulate or replace texels

# Filter Modes



Example:

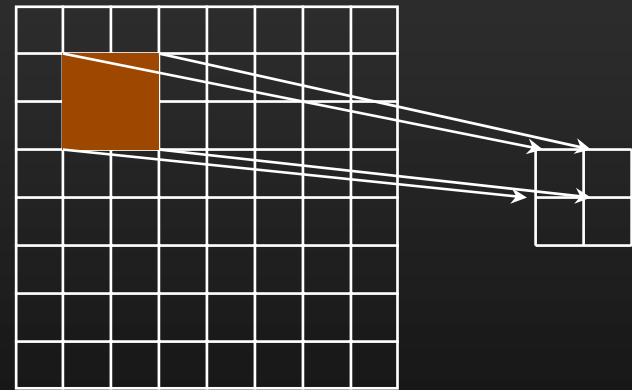
```
glTexParameteri( target, type, mode );
```



Texture

Polygon

Magnification



Texture

Polygon

Minification

# Mipmapped Textures



- Mipmap allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition  
`glTexImage*D( GL_TEXTURE_*D, level, ... )`
- GLU mipmap builder routines  
`gluBuild*DMipmaps( ... )`
- OpenGL 1.2 introduces advanced LOD controls



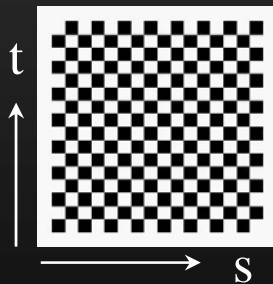
# Wrapping Mode



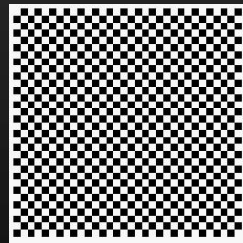
- **Example:**

```
glTexParameteri( GL_TEXTURE_2D,  
                 GL_TEXTURE_WRAP_S, GL_CLAMP )
```

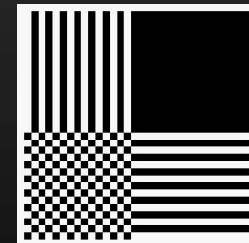
```
glTexParameteri( GL_TEXTURE_2D,  
                 GL_TEXTURE_WRAP_T, GL_REPEAT )
```



texture



GL\_REPEAT  
wrapping



GL\_CLAMP  
wrapping

# Texture Functions



- Controls how texture is applied

```
glTexEnv{f|v}[v]( GL_TEXTURE_ENV, prop, param )
```

- *GL\_TEXTURE\_ENV\_MODE* modes

- *GL\_MODULATE*

- *GL\_BLEND*

- *GL\_REPLACE*

- Set blend color with

```
GL_TEXTURE_ENV_COLOR
```

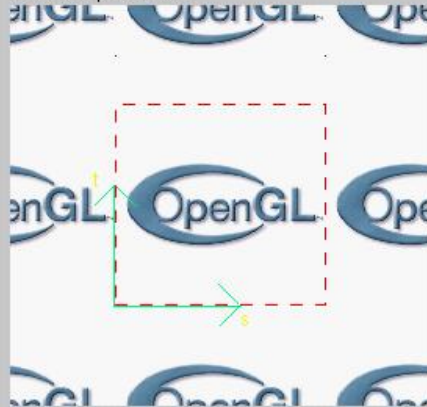
# Tutorial: Texture



Screen-space view



Texture-space view



Command manipulation window

```
GLfloat border_color[] = { 1.00 , 0.00 , 0.00 , 1.00 };
GLfloat env_color[] = { 0.00 , 1.00 , 0.00 , 1.00 };

glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, env_color);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

glEnable(GL_TEXTURE_2D);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, w, h, GL_RGB, GL_UNSIGNED_BYTE, image);

glColor4f( 0.60 , 0.60 , 0.60 , 1.00 );
glBegin(GL_POLYGON);
glTexCoord2f( 0.0 , 0.0 ); glVertex3f( -1.0 , -1.0 , 0.0 );
glTexCoord2f( 1.0 , 0.0 ); glVertex3f( 1.0 , -1.0 , 0.0 );
glTexCoord2f( 1.0 , 1.0 ); glVertex3f( 1.0 , 1.0 , 0.0 );
glTexCoord2f( 0.0 , 1.0 ); glVertex3f( -1.0 , 1.0 , 0.0 );
glEnd();
```

Click on the arguments and move the mouse to modify values.

# Perspective Correction Hint



- **Texture coordinate and color interpolation**
  - either linearly in screen space
  - or using depth/perspective values (slower)
- **Noticeable for polygons “on edge”**

**`glHint( GL_PERSPECTIVE_CORRECTION_HINT, hint )`**

where *hint* is one of

- **`GL_DONT_CARE`**
- **`GL_NICEST`**
- **`GL_FASTEST`**

# Is There Room for a Texture?



- **Query largest dimension of texture image**
  - typically largest square texture
  - doesn't consider internal format size

```
glGetIntegerv( GL_MAX_TEXTURE_SIZE, &size )
```

- **Texture proxy**
  - will memory accommodate requested texture size?
  - no image specified; placeholder
  - if texture won't fit, texture state variables set to 0
    - doesn't know about other textures
    - only considers whether this one texture will fit all of memory

# Texture Residency



- **Working set of textures**
  - high-performance, usually hardware accelerated
  - textures must be in texture objects
  - a texture in the *working set* is resident
  - for residency of current texture, check `GL_TEXTURE_RESIDENT` state
- **If too many textures, not all are resident**
  - can set priority to have some kicked out first
  - establish 0.0 to 1.0 priorities for texture objects

# Mini Projet



- **Systeme de base donn **
  - Systeme “complet”
  - Systeme “donn ”
- **3 T ches**
  - Viewer avec la souris
  - Chargement et affichage des textures
  - Selection et animation d’un objet

# MiniRV



- **Systeme donne :**
  - Chargement d'une scene par un « parser » du format obj/mtl
  - « cadre general » OpenGL/fenêtres etc.
- **TP décrit dans tp.doc sur le site**



# Tâche 1 : Viewer



- **Les fonctionnalités doivent être les suivantes :**
  - Bouton droit + déplacement : rotation
  - Bouton gauche + déplacement : translations 2D dans le plan de vue
  - SHIFT + Bouton gauche + déplacement (haut/bas) : zoom
- **Example**

# Tâche 1 : Viewer



Actuellement tout est géré par (RvCamera.cpp) :

```
RvViewer::processMouseEvents(button,button_state,modifiers,x,y,type);
```

Appelée depuis le callback "bouton souris" de GLUT :

```
void myGlutMouse(int button, int button_state, int x, int y )
{
    viewer-> processMouseEvents(button, button_state, glutGetModifiers(),
    x, y);
}
```

et une fonction de mise à jour :

```
void RvViewer::update(int mousex,int mousey);
```

appelée depuis le callback "mouvement souris" de GLUT

```
void myGlutMotion(int x, int y )
{
    viewer->update(x,y);
}
```

# Tâche 1 : Viewer



Il faudra compléter les méthodes de RvExaminerViewer (dans RvCamera.h) :

- begin/end rotate
- begin/end translate
- begin/end zoom

ainsi que

- rotate
- translate
- zoom

permettant de gérer la camera. Les endroits à changer dans le code sont indiqués par le symbole :

//....

# Tâche 2 : ajouter les textures



- Actuellement textures lu par le « parser »
  - Passés au programme mais pas traités
    - Etat actuel
    - Etat final

# Tâche 2 : ajouter les textures



- **Il faudra modifier les fonctions de traitement des textures**
  - Ajouter les champs
  - Compléter les fonctions d'initialisation
  - Gérer l'état OpenGL correctement pour les textures

# Tâche 3 : Sélection et animation



- **Sélectionner un objet par tracé de rayons**
  - Envoyer un rayon dans la scène
  - Implémenter la routine d'intersection avec l'objet
- **Animer l'objet sélectionné**
  - Par des touches par exemple

# On-Line Resources



- <http://www.opengl.org>
  - start here; up to date specification and lots of sample code
- <news:comp.graphics.api.opengl>
- <http://www.sgi.com/software/opengl>
- <http://www.mesa3d.org/>
  - Brian Paul's Mesa 3D
- <http://www.cs.utah.edu/~narobins/opengl.html>
  - very special thanks to Nate Robins for the OpenGL Tutors
  - source code for tutors available here!

# Books



- **OpenGL Programming Guide, 3<sup>rd</sup> Edition**
- **OpenGL Reference Manual, 3<sup>rd</sup> Edition**
- **OpenGL Programming for the X Window System**
  - includes many GLUT examples
- **Interactive Computer Graphics: A top-down approach with OpenGL, 2<sup>nd</sup> Edition**





**FIN**



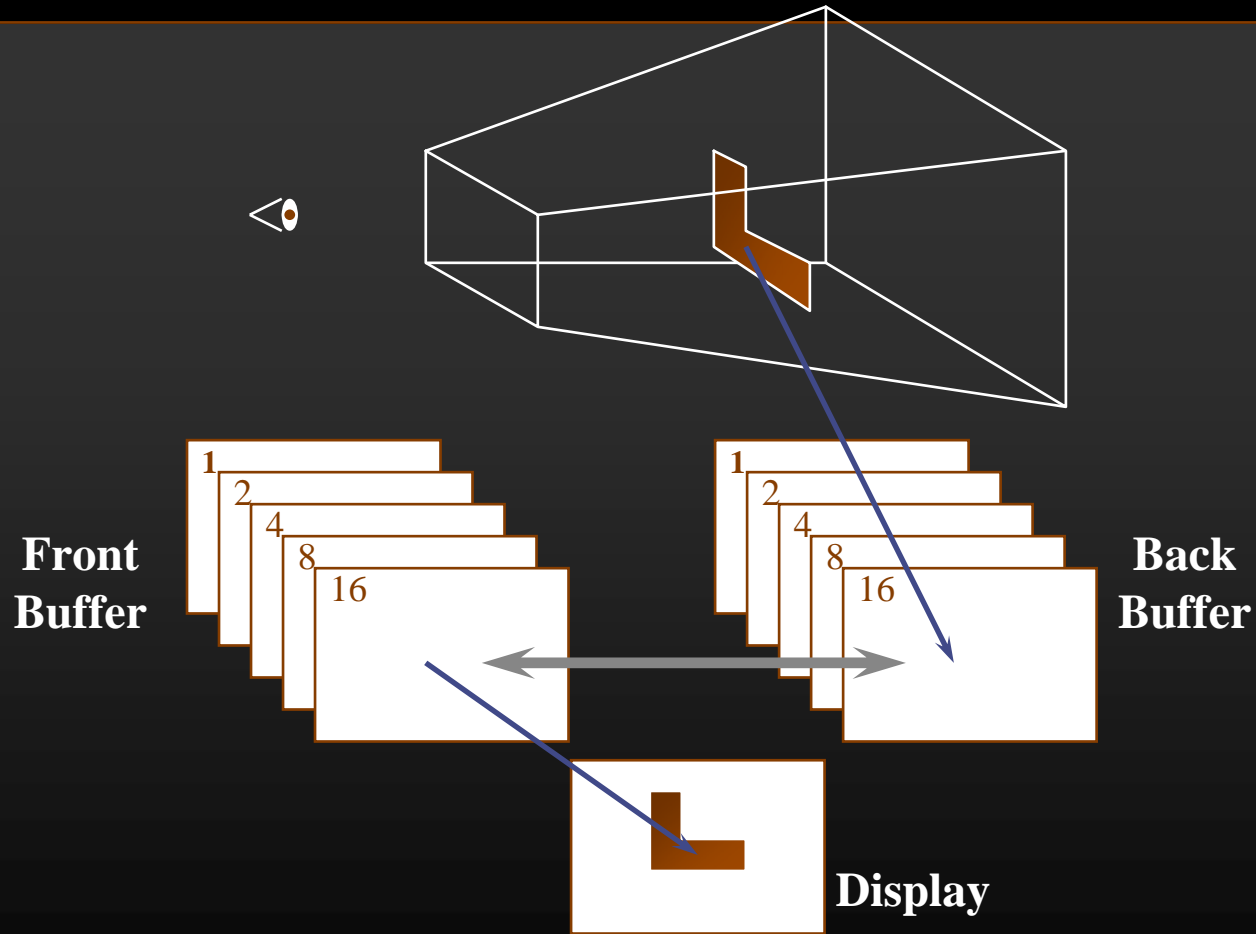
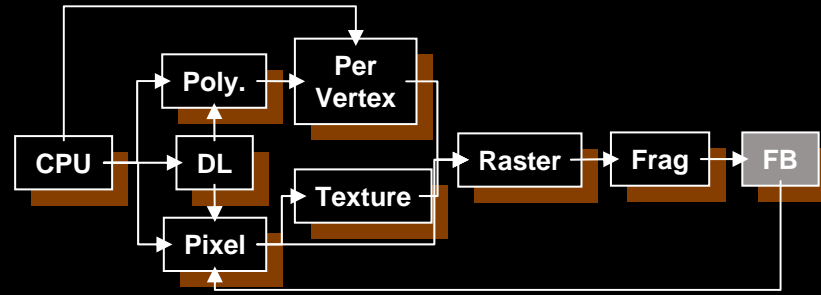
# Animation and Depth Buffering

# Animation and Depth Buffering



- Discuss double buffering and animation
- Discuss hidden surface removal using the depth buffer

# Double Buffering



# Animation Using Double Buffering



- ① Request a double buffered color buffer

```
glutInitDisplayMode( GLUT_RGB /  
GLUT_DOUBLE );
```

- ② Clear color buffer

```
glClear( GL_COLOR_BUFFER_BIT );
```

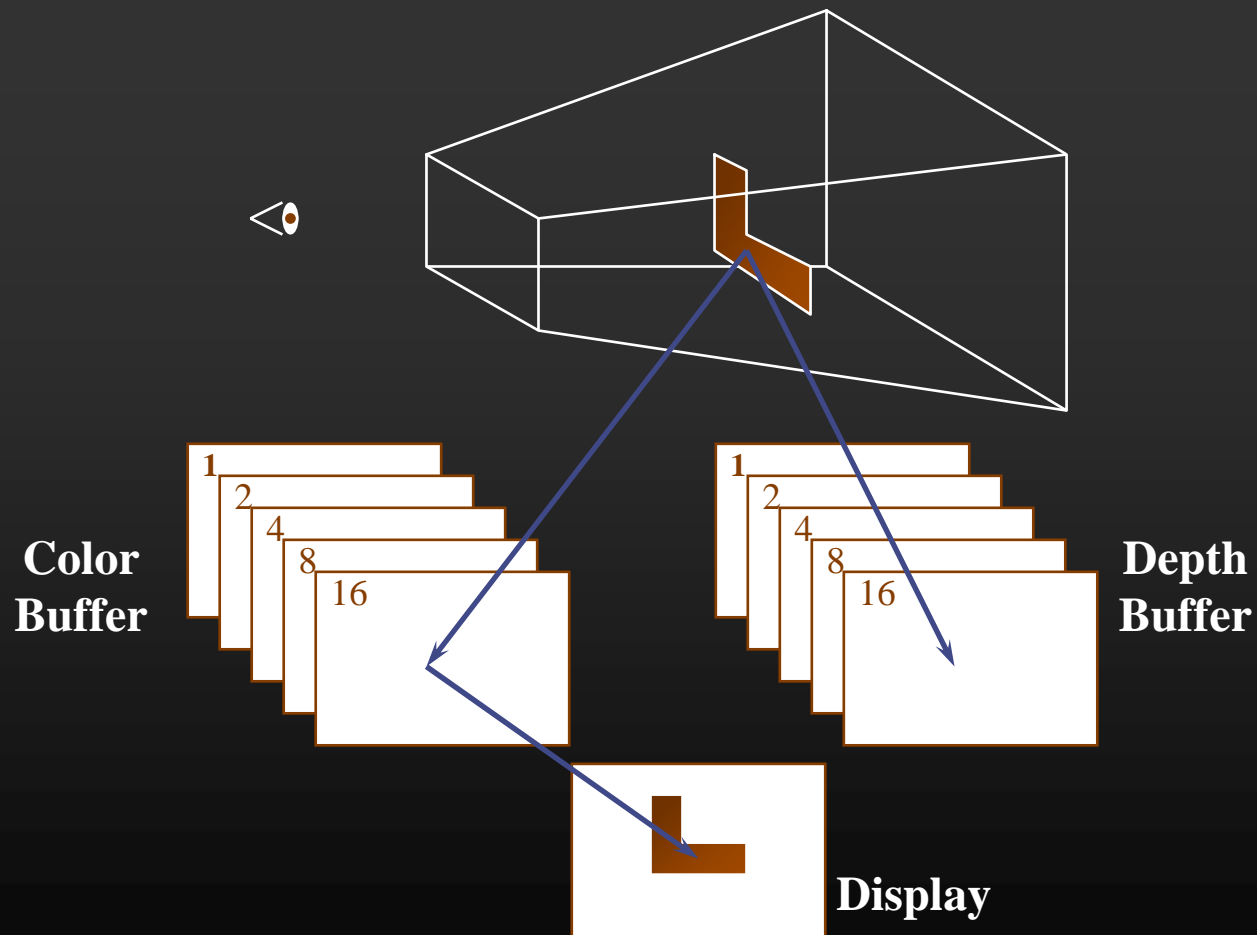
- ③ Render scene

- ④ Request swap of front and back buffers

```
glutSwapBuffers();
```

- Repeat steps 2 - 4 for animation

# Depth Buffering and Hidden Surface Removal



# Depth Buffering Using OpenGL



## ① Request a depth buffer

```
glutInitDisplayMode( GLUT_RGB /  
GLUT_DOUBLE / GLUT_DEPTH );
```

## ② Enable depth buffering

```
glEnable( GL_DEPTH_TEST );
```

## ③ Clear color and depth buffers

```
glClear( GL_COLOR_BUFFER_BIT /  
GL_DEPTH_BUFFER_BIT );
```

## ④ Render scene

## ⑤ Swap color buffers

# An Updated Program Template



```
void main( int argc, char** argv )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGB |
        GLUT_DOUBLE | GLUT_DEPTH );
    glutCreateWindow( "Tetrahedron" );
    init();
    glutIdleFunc( idle );
    glutDisplayFunc( display );
    glutMainLoop();
}
```



# An Updated Program Template (cont.)



```
void init( void )
{
    glClearColor( 0.0, 0.0, 1.0, 1.0 );
}
```

```
void idle( void )
{
    glutPostRedisplay();
}
```

# An Updated Program Template (cont.)



```
void drawScene( void )
{
    GLfloat vertices[] = { ... };
    GLfloat colors[] = { ... };
    glClear( GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT );
    glBegin( GL_TRIANGLE_STRIP );
    /* calls to glColor*() and glVertex*() */
    glEnd();
    glutSwapBuffers();
}
```



# Advanced OpenGL Topics



# Advanced OpenGL Topics



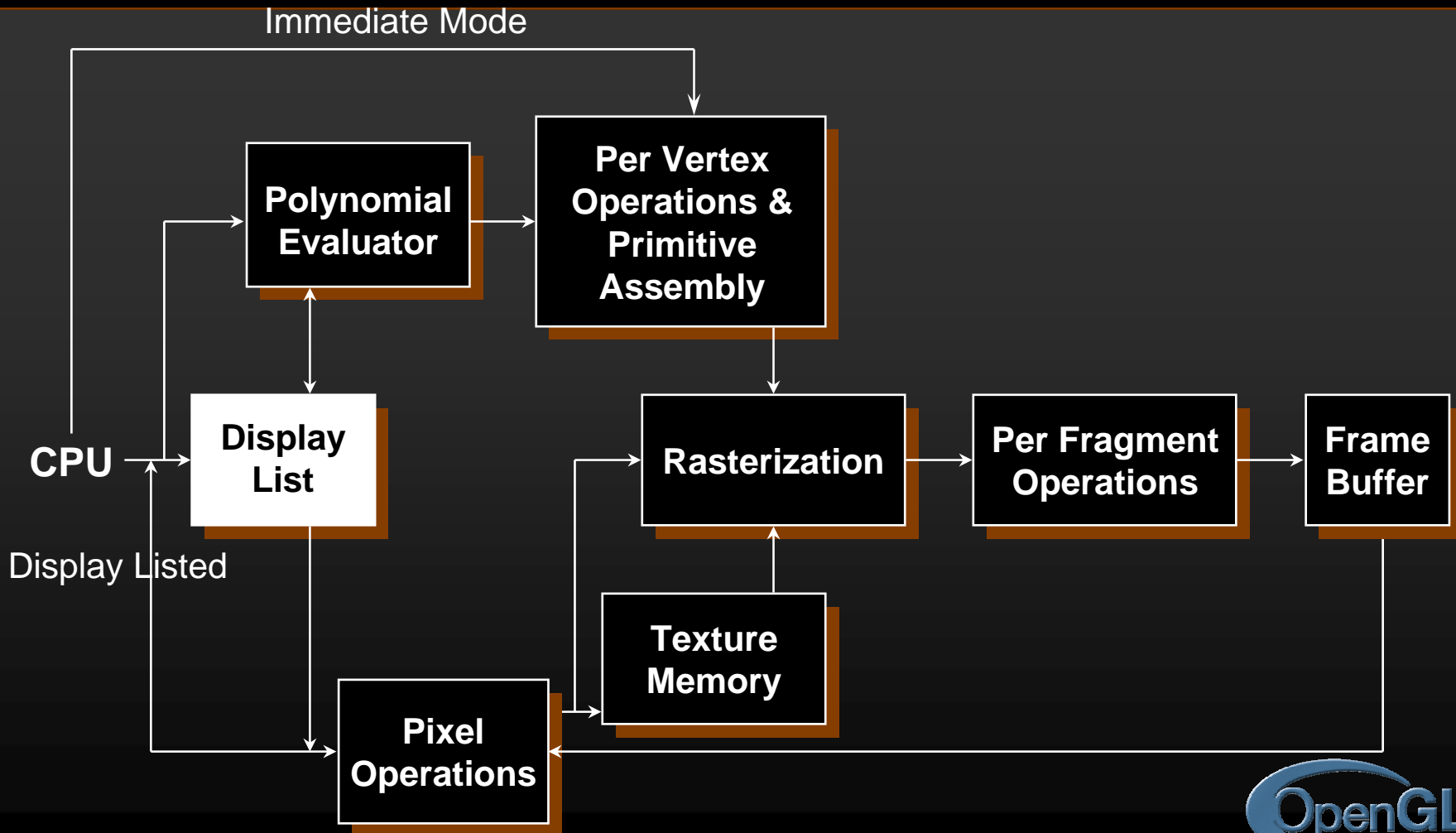
- **Display Lists and Vertex Arrays**
- **Alpha Blending and Antialiasing**
- **Using the Accumulation Buffer**
- **Fog**
- **Feedback & Selection**
- **Fragment Tests and Operations**
- **Using the Stencil Buffer**

# Immediate Mode versus Display Listed Rendering

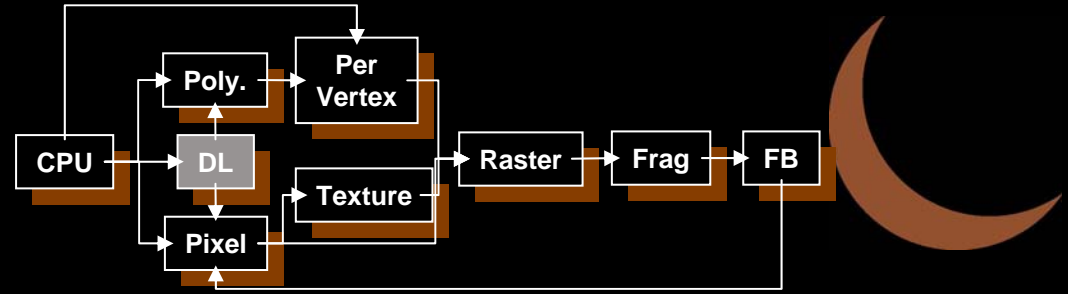


- **Immediate Mode Graphics**
  - Primitives are sent to pipeline and display right away
  - No memory of graphical entities
- **Display Listed Graphics**
  - Primitives placed in display lists
  - Display lists kept on graphics server
  - Can be redisplayed with different state
  - Can be shared among OpenGL graphics contexts

# Immediate Mode versus Display Lists



# Display Lists



- **Creating a display list**

```
GLuint id;
void init( void )
{
    id = glGenLists( 1 );
    glNewList( id, GL_COMPILE );
    /* other OpenGL routines */
    glEndList();
}
```

- **Call a created list**

```
void display( void )
{
    glCallList( id );
}
```

# Display Lists



- **Not all OpenGL routines can be stored in display lists**
- **State changes persist, even after a display list is finished**
- **Display lists can call other display lists**
- **Display lists are not editable, but you can fake it**
  - make a list (A) which calls other lists (B, C, and D)
  - delete and replace B, C, and D, as needed

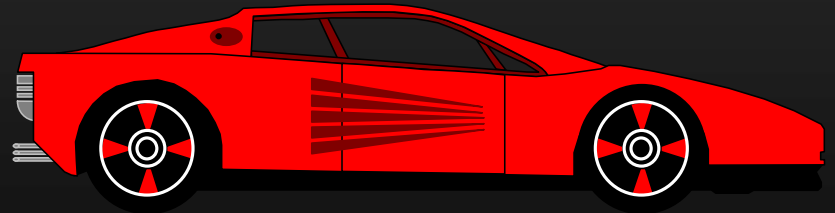


# Display Lists and Hierarchy



- **Consider model of a car**
  - Create display list for chassis
  - Create display list for wheel

```
glNewList( CAR, GL_COMPILE );  
  glCallList( CHASSIS );  
  glTranslatef( ... );  
  glCallList( WHEEL );  
  glTranslatef( ... );  
  glCallList( WHEEL );  
  ...  
glEndList();
```

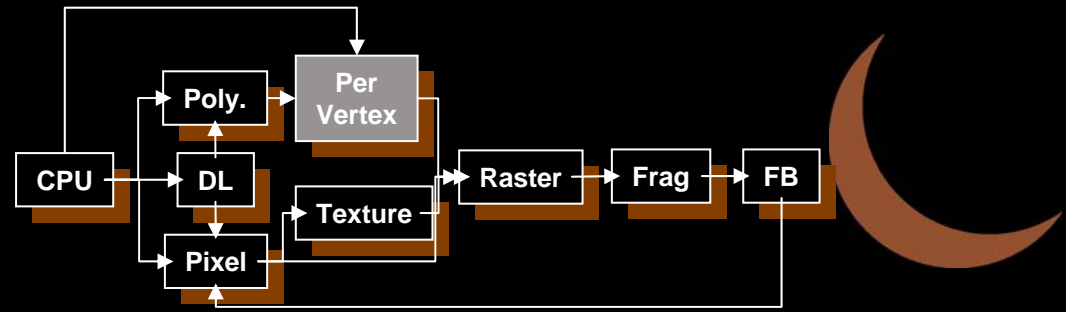


# Advanced Primitives



- **Vertex Arrays**
- **Bernstein Polynomial Evaluators**
  - basis for GLU NURBS
    - NURBS (Non-Uniform Rational B-Splines)
- **GLU Quadric Objects**
  - sphere
  - cylinder (or cone)
  - disk (circle)

# Vertex Arrays



- **Pass arrays of vertices, colors, etc. to OpenGL in a large chunk**

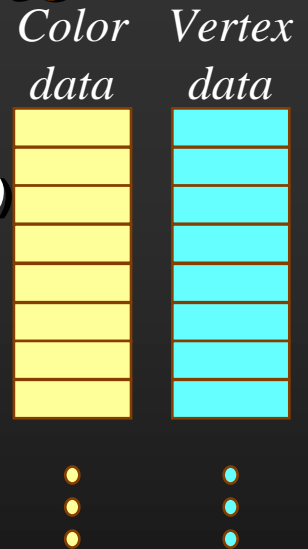
```
glVertexPointer( 3, GL_FLOAT, 0, coords )
```

```
glColorPointer( 4, GL_FLOAT, 0, colors )
```

```
glEnableClientState( GL_VERTEX_ARRAY )
```

```
glEnableClientState( GL_COLOR_ARRAY )
```

```
glDrawArrays( GL_TRIANGLE_STRIP, 0, numVerts );
```



- **All active arrays are used in rendering**



# Why use Display Lists or Vertex Arrays?



- **May provide better performance than immediate mode rendering**
- **Display lists can be shared between multiple OpenGL context**
  - reduce memory usage for multi-context applications
- **Vertex arrays may format data for better memory access**

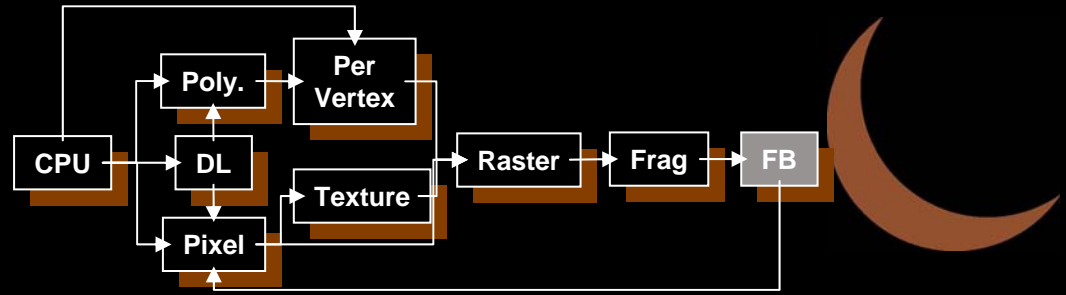
# Alpha: the 4<sup>th</sup> Color Component



- **Measure of Opacity**
  - simulate translucent objects
    - glass, water, etc.
  - composite images
  - antialiasing
  - ignored if blending is not enabled

```
glEnable( GL_BLEND )
```

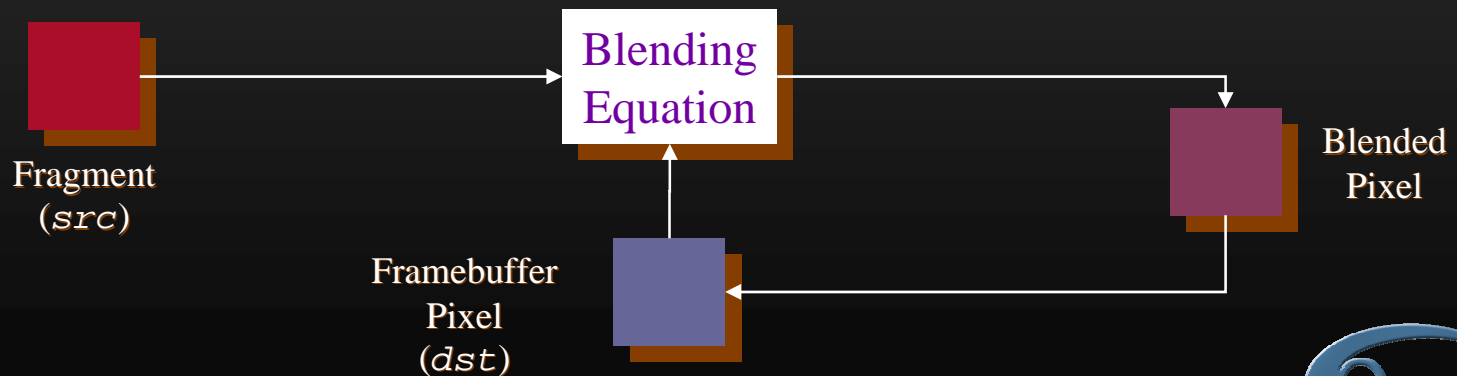
# Blending



- Combine pixels with what's already in the framebuffer

`glBlendFunc( src, dst )`

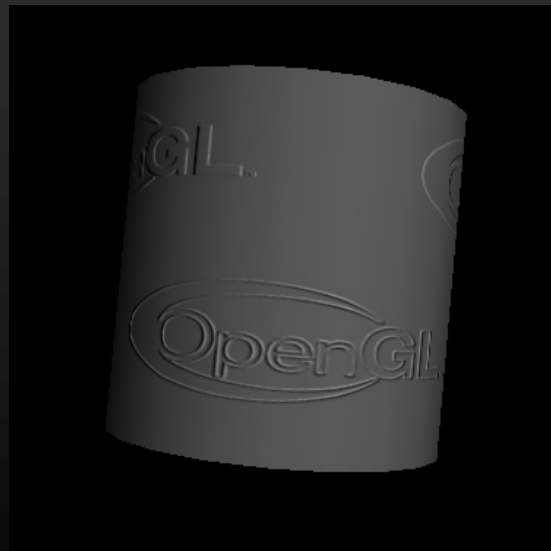
$$\vec{C}_r = src \vec{C}_f + dst \vec{C}_p$$



# Multi-pass Rendering



- **Blending allows results from multiple drawing passes to be combined together**
  - enables more complex rendering algorithms



Example of bump-mapping  
done with a multi-pass  
OpenGL algorithm

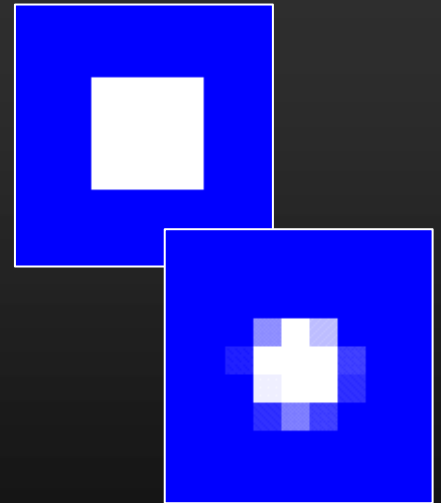
# Antialiasing



- Removing the Jaggies

`glEnable( mode )`

- `GL_POINT_SMOOTH`
- `GL_LINE_SMOOTH`
- `GL_POLYGON_SMOOTH`
- alpha value computed by computing sub-pixel coverage
- available in both RGBA and colormap modes





# Accumulation Buffer



- **Problems of compositing into color buffers**
  - limited color resolution
    - clamping
    - loss of accuracy
  - Accumulation buffer acts as a “floating point” color buffer
    - accumulate into accumulation buffer
    - transfer results to frame buffer

# Accessing Accumulation Buffer



**glAccum( op, value )**

- operations
  - within the accumulation buffer: *GL\_ADD*, *GL\_MULT*
  - from read buffer: *GL\_ACCUM*, *GL\_LOAD*
  - transfer back to write buffer: *GL\_RETURN*
- **glAccum(*GL\_ACCUM*, 0.5)** multiplies each value in write buffer by 0.5 and adds to accumulation buffer

# Accumulation Buffer Applications

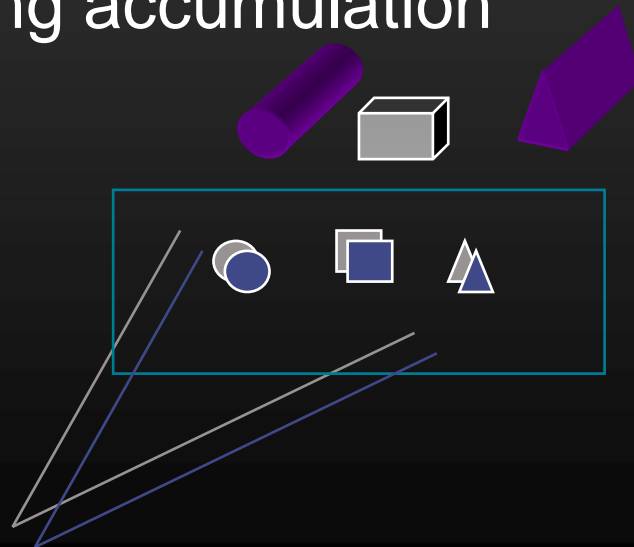


- **Compositing**
- **Full Scene Antialiasing**
- **Depth of Field**
- **Filtering**
- **Motion Blur**

# Full Scene Antialiasing : *Jittering the view*



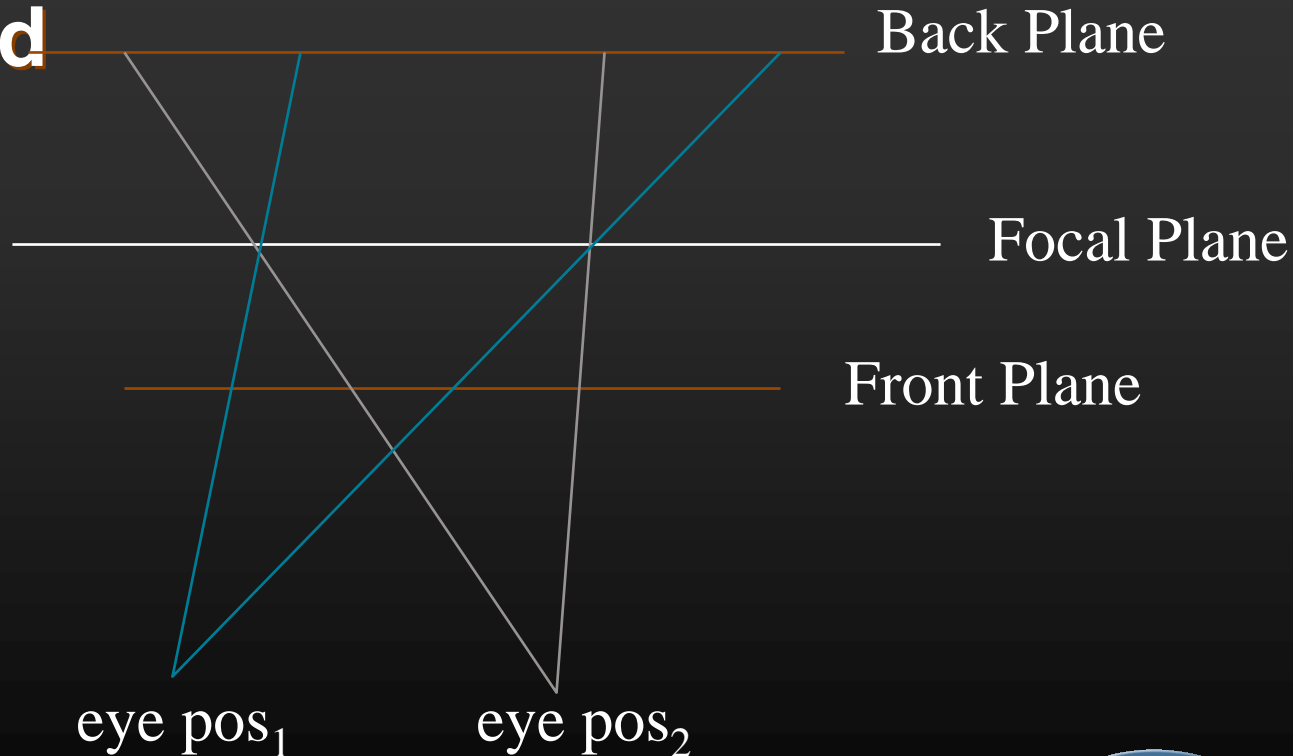
- Each time we move the viewer, the image shifts
  - Different aliasing artifacts in each image
  - Averaging images using accumulation buffer averages out these artifacts



# Depth of Focus : *Keeping a Plane in Focus*



- Jitter the viewer to keep one plane unchanged



# Fog



`glFog( property, value )`

- **Depth Cueing**

- Specify a range for a linear fog ramp
  - `GL_FOG_LINEAR`

- **Environmental effects**

- Simulate more realistic fog
  - `GL_FOG_EXP`
  - `GL_FOG_EXP2`

# Fog Tutorial




Fog

Fog equation

$$f = \frac{\text{end} - z}{\text{end} - \text{start}}$$

z is the distance in eye coordinates from origin to fragment being fogged.

Screen-space view



Command manipulation window

```
GLfloat color[4] = { 0.70 , 0.70 , 0.70 , 1.00 };
glFogfv(GL_FOG_COLOR, color);
glFogf(GL_FOG_START, 0.50 );
glFogf(GL_FOG_END, 2.00 );
glFogi(GL_FOG_MODE, GL_LINEAR);
```

Click on the arguments and move the mouse to modify values.

# Feedback Mode



- **Transformed vertex data is returned to the application, not rendered**
  - useful to determine which primitives will make it to the screen
- **Need to specify a feedback buffer**  
`glFeedbackBuffer( size, type, buffer )`
- **Select feedback mode for rendering**  
`glRenderMode( GL_FEEDBACK )`



# Selection Mode



- Method to determine which primitives are inside the viewing volume
- Need to set up a buffer to have results returned to you

`glSelectBuffer( size, buffer )`

- Select selection mode for rendering

`glRenderMode( GL_SELECT )`

# Selection Mode (cont.)



- **To identify a primitive, give it a name**
  - “names” are just integer values, not strings
- **Names are stack based**
  - allows for hierarchies of primitives
- **Selection Name Routines**

```
glLoadName( name )   glPushName( name )  
glInitNames( )
```

# Picking



- **Picking is a special case of selection**
- **Programming steps**
  - restrict “drawing” to small region near pointer  
USE `gluPickMatrix()` on projection matrix
  - enter selection mode; re-render scene
  - primitives drawn near cursor cause hits
  - exit selection; analyze hit records

# Picking Template



```
glutMouseFunc( pickMe );

void pickMe( int button, int state, int x, int y )
{
    GLuint nameBuffer[256];
    GLint hits;
    GLint myViewport[4];
    if (button != GLUT_LEFT_BUTTON ||
        state != GLUT_DOWN) return;
    glGetIntegerv( GL_VIEWPORT, myViewport );
    glSelectBuffer( 256, nameBuffer );
    (void) glRenderMode( GL_SELECT );
    glInitNames();
}
```

# Picking Template (cont.)



```
glMatrixMode( GL_PROJECTION );
glPushMatrix();
glLoadIdentity();
gluPickMatrix( (GLdouble) x, (GLdouble)
    (myViewport[3]-y), 5.0, 5.0, myViewport );
/*  gluPerspective or glOrtho or other projection */
glPushName( 1 );
/*  draw something */
glLoadName( 2 );
/*  draw something else ... continue ... */
```

# Picking Template (cont.)



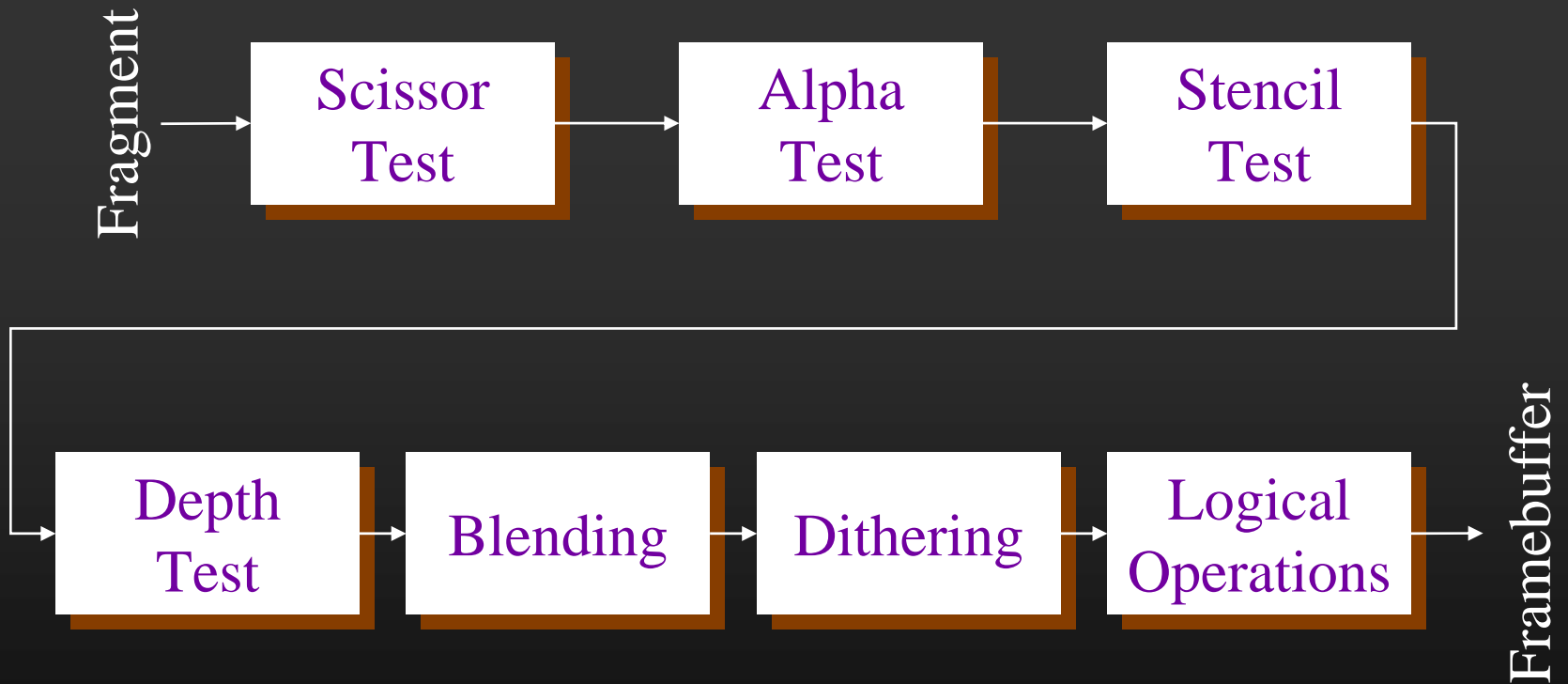
```
glMatrixMode( GL_PROJECTION );  
glPopMatrix();  
hits = glRenderMode( GL_RENDER );  
/*   process nameBuffer   */  
}
```

# Picking Ideas



- **For OpenGL Picking Mechanism**
  - only render what is pickable (e.g., don't clear screen!)
  - use an “invisible” filled rectangle, instead of text
  - if several primitives drawn in picking region, hard to use z values to distinguish which primitive is “on top”
- **Alternatives to Standard Mechanism**
  - color or stencil tricks (for example, use `glReadPixels()` to obtain pixel value from back buffer)

# Getting to the Framebuffer





# Scissor Box

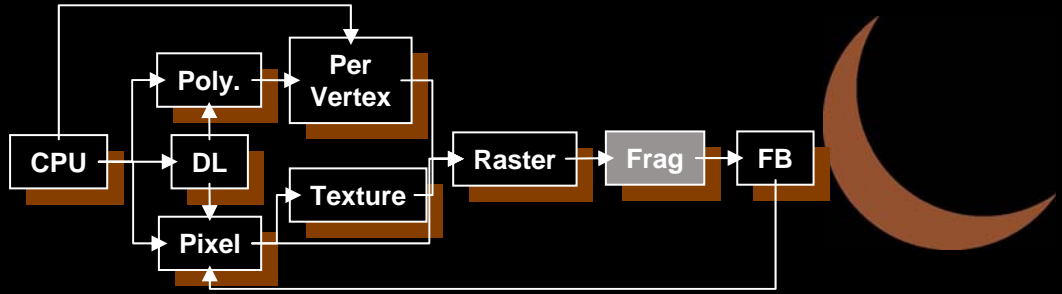


- **Additional Clipping Test**

**`glScissor( x, y, w, h )`**

- any fragments outside of box are clipped
- useful for updating a small section of a viewport
  - affects `glClear( )` operations

# Alpha Test



- Reject pixels based on their alpha value

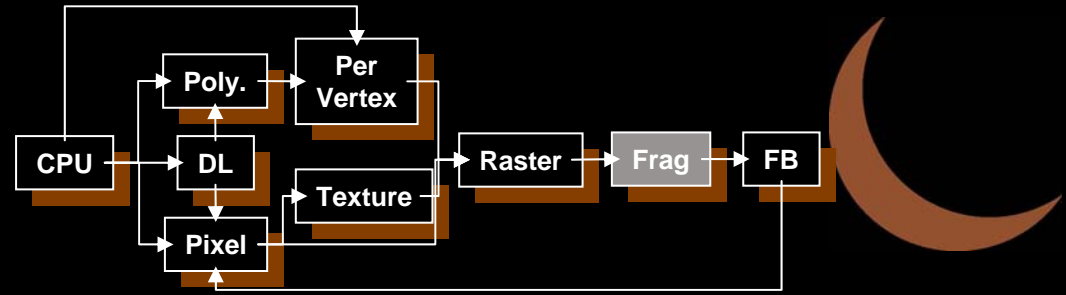
`glAlphaFunc( func, value )`

`glEnable( GL_ALPHA_TEST )`

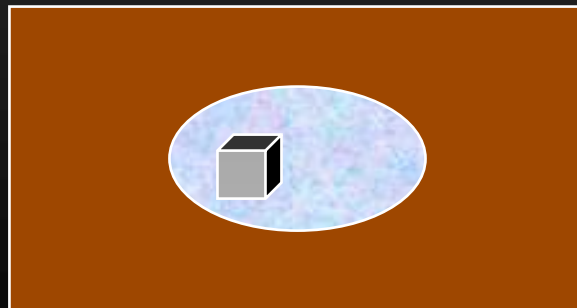
- use alpha as a mask in textures



# Stencil Buffer



- **Used to control drawing based on values in the stencil buffer**
  - Fragments that fail the stencil test are not drawn
  - Example: create a mask in stencil buffer and draw only objects not in mask area



# Controlling Stencil Buffer



**glStencilFunc( *func*, *ref*, *mask* )**

- compare value in buffer with **ref** using **func**
- only applied for bits in **mask** which are 1
- **func** is one of standard comparison functions

**glStencilOp( *fail*, *zfail*, *zpass* )**

- Allows changes in stencil buffer based on passing or failing stencil and depth tests: **GL\_KEEP**, **GL\_INCR**

# Creating a Mask



```
glInitDisplayMode( ...|GLUT_STENCIL|... );  
glEnable( GL_STENCIL_TEST );  
glClearStencil( 0x1 );
```

```
glStencilFunc( GL_ALWAYS, 0x1, 0x1 );  
glStencilOp( GL_REPLACE, GL_REPLACE,  
            GL_REPLACE );
```

- *draw mask*

# Using Stencil Mask



```
glStencilFunc( GL_EQUAL, 0x1, 0x1 )
```

- **draw objects where stencil = 1**

```
glStencilFunc( GL_NOT_EQUAL, 0x1, 0x1 );
```

```
glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );
```

- **draw objects where stencil != 1**

# Dithering



```
glEnable( GL_DITHER )
```

- **Dither colors for better looking results**
  - Used to simulate more available colors

# Logical Operations on Pixels



- Combine pixels using bitwise logical operations

`glLogicOp( mode )`

- Common modes
  - `GL_XOR`
  - `GL_AND`



# Advanced Imaging



- **Imaging Subset**

- Only available if `GL_ARB_imaging` defined
  - Color matrix
  - Convolutions
  - Color tables
  - Histogram
  - MinMax
  - Advanced Blending



**Diapos originales :**

**Dave Shreiner**

**Ed Angel**

**Vicki Shreiner**



# On-Line Resources



- <http://www.opengl.org>
  - start here; up to date specification and lots of sample code
- <news:comp.graphics.api.opengl>
- <http://www.sgi.com/software/opengl>
- <http://www.mesa3d.org/>
  - Brian Paul's Mesa 3D
- <http://www.cs.utah.edu/~narobins/opengl.html>
  - very special thanks to Nate Robins for the OpenGL Tutors
  - source code for tutors available here!

# Books



- **OpenGL Programming Guide, 3<sup>rd</sup> Edition**
- **OpenGL Reference Manual, 3<sup>rd</sup> Edition**
- **OpenGL Programming for the X Window System**
  - includes many GLUT examples
- **Interactive Computer Graphics: A top-down approach with OpenGL, 2<sup>nd</sup> Edition**

# Additional Clipping Planes



- **At least 6 more clipping planes available**
- **Good for cross-sections**
- **Modelview matrix moves clipping plane**
- $Ax + By + Cz + D < 0$  **clipped**
- `glEnable( GL_CLIP_PLANEi )`
- `glClipPlane( GL_CLIP_PLANEi, GLdouble* coeff )`