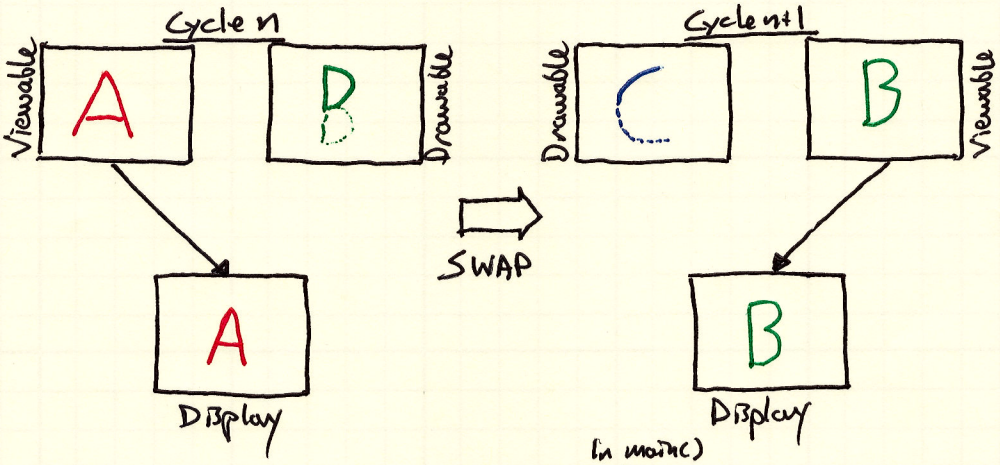
Outcomes

- Understand the basic concepts of double buffering
- Understand how viewing transformations work
- Be able to invert and manipulate the view with either
  - Viewing transform
  - Modeling transforms
- Understand the basic concepts of collision detection
- Understand how selection and picking work in OpenGL

Double Buffering

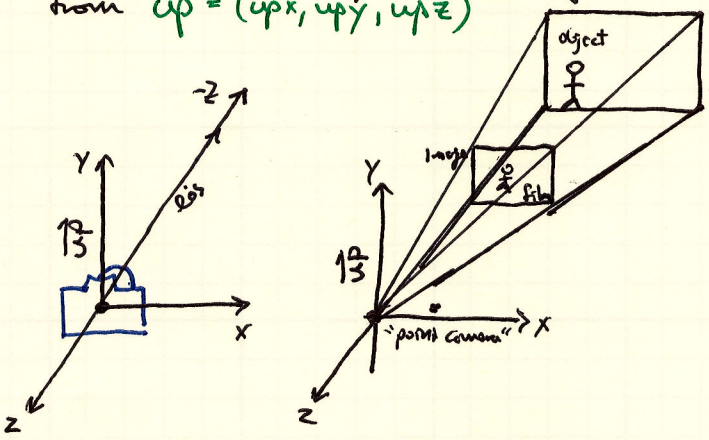
- Critical for smooth animations
- Color Buffers:
  - Spaces in GPU memory used to store RGBA values for raster pixels
- Space for 2 Color Buffers are provided in a double-buffered system
  - Viewable Buffer: holds frame currently displayed (aka front or main buffer)
  - Drawable Buffer: Location where the next frame is being drawn (aka back or swap buffer)
- When rendering is complete in the drawable buffer, the buffers are swapped.
- Double buffering greatly improves the quality of the animation. This is done by hiding the transition between successive frames from the viewer



- Double Buffering can be enabled by: calling glutDisplayMode() (in main) with GLUT_DOUBLE (instead of GLUT_SINGLE) as an of the arguments
- Double Buffering, typically is implemented in the GPU, both buffers are in VRAM
- Note this can cause visual artifacts if the draw-and-swap loop renders out-of-sync w/ the refresh rate of the display
  - Screen tearing - i.e. GPU draws frames @ 80 fps, display refreshes at 60 Hz. GPU takes 0.0125 secs to render but the monitor takes 0.0167 secs. This results in the monitor starting to show a new frame when only 75% (0.0125/0.0167) of the previous has been rendered.
    - Avoid by imposing vertical synchronization (v-sync) or triple-buffering

Viewing Transformation

- In OpenGL, the viewing transformation is gluLookAt()
  - Functions to arrange OpenGL's imaginary camera
- Default orientation: -Z direction aligned along +Y
- gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)
  - Moves camera to location eye = (eyex, eyey, eyez)
  - Points it at center = (centerx, centery, centerz)
  - Rotates about line of sight (line joining eye + center) and determines up direction from up = (upx, upy, upz)



- GluLookAt gives us the convenience to move about the scene, rather than having to move the scene into the view.
- gluLookAT() does not change the Frustum, only its placement and alignment
- Note: $\vec{up}$ is the actual up direction
- Collectively the ModelView Transformations are:
  - glTranslatef()
  - glRotatef()
  - glScalef()
  - gluLookAt()

Ex:

Simulating Viewing Transformations

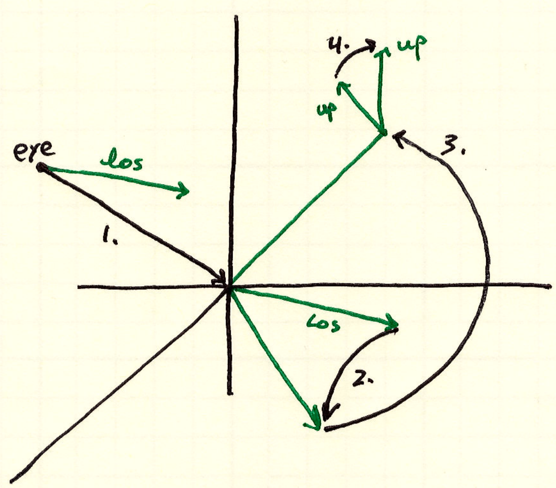
- gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)

  is equivalent to the modeling transformations: (fixed axis rotation)

  ```
  glRotatef(B, 0.0, 0.0, 1.0)
  glRotatef(A, wx, wy, wz)
  glTranslatef(-eyex, -eyey, -eyez)
  ```
  (the two glRotatef calls are dependent on line of sight)

  it is also equivalent to (in general) is

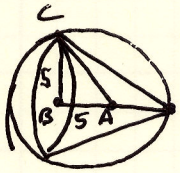
  ```
  glRotatef(-γ, 0.0, 0.0, 1.0)
  glRotatef(-β, 0.0, 1.0, 0.0)
  glRotatef(-α, 1.0, 0.0, 0.0)
  glTranslate(-eyex, -eyey, -eyez)
  ```
  to restore →
  1. glTranslate(-eyex, -eyey, -eyez)
  2. glRotate(-α, 1.0, 0.0, 0.0)
  3. glRotate(-β, 0.0, 1.0, 0.0)
  4. glRotate(-γ, 0.0, 0.0, 1.0)

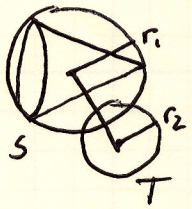


- The angles α, β, γ are the camera's euler angles
  - They determine its orientation

Collision Detection

- The process of detecting if two objects have / will intersect
- Typically done in an approximate way to reduce complexity of calculations
- Ex: check spheres intersection



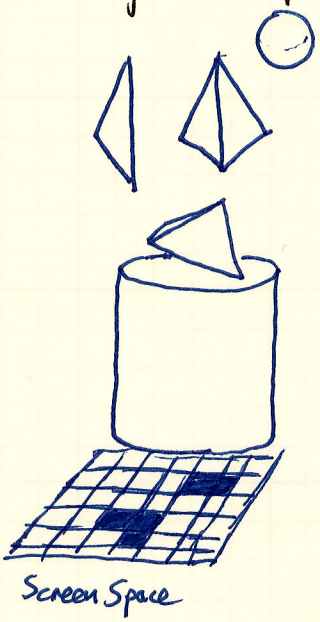


$$|AC| = \sqrt{|AB|^2 + |BC|^2}$$

To detect collision we check the bounding spheres, rather than the objects themselves

Selection and Picking: Selection

- Selection and picking provide the capability to select and manipulate an object via interaction in the 3D world
- Picking is the act of identifying which object belongs to a pixel on the screen.
  - This is a relatively complex process, given the method by which objects are rasterized.
  - Luckily, OpenGL provides a means to facilitate this process



- Selection:
  - Allows the user to specify a viewing volume and to then select the objects which intersect this volume.
  - Process of using Selection

1.) Create the hit buffer

```
glSelectBuffer(size, buffer) : size, buffer size in bytes, buffer array
```

2.) Enter Selection Render mode

```
glRenderMode(GL_SELECT)
```

3.) Store the current matrix on the stack, setup the new projection for the selection volume, and re-enter model view matrix and set the identity

```
glMatrixMode(GL_PROJECTION)
glPushMatrix()
glLoadIdentity()
// set projection w/ glOrth, glFrustum, etc...
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
```

4.) Initialize and push an empty name stack

```
glInitNames()
glPushName(0)
```

5.) Render, like normal (though nothing is drawn), then shift back to Render mode

```
hits = glRenderMode(GL_RENDER)
```

Selection and Picking: Picking

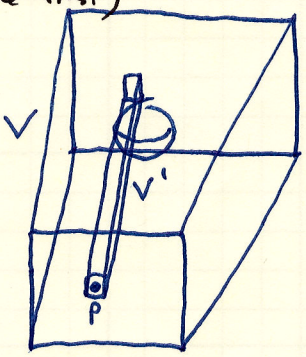
6.) Finally Reset Projection matrix

```
glMatrixMode(GL_PROJECTION)
glPopMatrix()
glMatrixMode(GL_MODELVIEW)
```

- A Hit record is recorded in the buffer, when:
  - A name stack manipulation or glRenderMode() command is encountered, and
  - A hit has occurred (primitive drawn in the volume)
- Each hit record contains the follow fields (in order)
  1. Number of names in name stack
  2. Minimum z-value of vertices belonging to primitives that hit the selection volume
  3. Max z-value of vertices of primitives that hit the selection volume
  4. Sequence of names in name stack (bottom one first)

Picking:

- Selection plus some OpenGL help
- V - Viewing Frustum defined by projection statement
  P - Point selected in OpenGL viewing window
  V' - Viewing volume in V acting as selection volume



- V' can be determined using gluPickMatrix()

```
glLoadIdentity()
gluPickMatrix(pickX, pickY, width, height, viewport)
glFrustum(); or gluPerspective(); or glOrtho()

glGetIntegerv(GL_VIEW_PORT, viewport)
```

pickX, pickY – center of rectangle of width x height in the display

viewport – integer array of the x, y coordinates and w/h of the viewport, set by calling glGetIntegerV()

CS 4458 Lecture 07

For Next Time

1. Review previous lectures
2. Read Ch 5 sections 1-4
3. Start HW 02
4. Come to Class!

Additional Notes: