Outline: 9/28

- Python debugging
- Coordinate frames
- Clipping
- OpenGL intro and demo
Debugging Demo

- Small example
- Print statements
- Google
- Ask the whole class on Piazza
- Email me
Coordinate Systems

- Screen Coords

(0,0)
Coordinate Systems

- Screen Coords
- Viewport
- World Coords
Coordinate Systems

• Use transformation matrices to change coordinate systems
3D Graphics

- 2 major approaches
  - “Graphics Pipeline”/OpenGL (now)
  - Ray Casting/Tracing (~4 weeks)
3D Approaches

- Graphics Pipeline
- Project 3D → 2D
  - Transformations
- Polygon Rasterization
- Limited Effects/Tricks
- Fast (Real Time)
  - Hardware Accelerated
- Ray Casting/Tracing
- Simulation of Eye/Camera
- Rendering of Solids
- Lots of effects (simulation)
- Slow
  - But getting faster
Approach

- map 3d vertices $\rightarrow$ screen
- process objects one at a time as they are being generated by the application

All step can be implemented in hardware on the graphics card
The Pipeline: Transformations

• Changes from one coordinate system to another
  • Object coordinates
  • world coordinates
  • Camera coordinates

vertices ➔ transformer
The Pipeline: Clipping

- Eliminate objects the camera can’t see.

Clipping

- Just as a real camera cannot “see” the whole world, the virtual camera can only see part of the world space:

  - Objects that are not within this volume are said to be clipped out of the scene.
Clipping
Clipping
The Pipeline: Projection

- Transformation of 3D objects to 2D
- Camera Coordinates to Screen Coordinates
- two ways: perspective and orthographic
Orthographic vs Perspective Projection

- Perspective
  - Vanishing Point
  - Foreshortening

- Orthographic
  - Parallel Projection
  - Distances Preserved
OpenGL is not OOP

OpenGL is not object oriented and its python libraries are auto generated so you get functions like this...

```python
glVertex3f
glVertex2i
glVertex3d
```
`glVertex3f(x, y, z)`

- GL library
- Function name
- 3 floats
OpenGL Primitives