Now Playing:

Coulibaly
Amadou & Mariam
from *Dimanche a Bamako*
Released August 2, 2005

Vertex Processing: Viewing

Rick Skarbez, Instructor
COMP 575
September 27, 2007

Announcements

- Programming Assignment 1 is due TONIGHT at 11:59pm
  - If you want to demo your program in person, I'd like to do it Friday afternoon if possible
  - Please contact me by email (TODAY) to set up a time

Submitting Programs

- Upload source and executable(s) (Windows or Mac) to digital dropbox on Blackboard
  - blackboard.unc.edu
- Include a document that lists
  - What optional components you did
  - Instructions for use
  - Any problems that you had, or components that do not work properly
- Please submit as a zip file with your name in the filename

Last Time

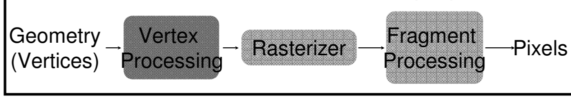
- Presented the functions needed for lighting and shading in OpenGL
- Demoed some lighting and shading functions in OpenGL
- Briefly discussed Non-Photorealistic Rendering (NPR)

Today

- Review the OpenGL pipeline
- Discuss viewing and how it applies to computer graphics

Rendering Pipeline

- OpenGL rendering works like an assembly line
  - Each stage of the pipeline performs a distinct function on the data flowing by
  - Each stage is applied to every vertex to determine its contribution to output pixels

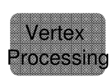


Vertex Processing

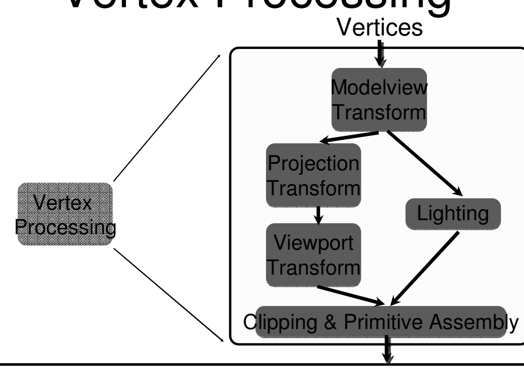
- The job of the vertex processing step is to take arbitrary input geometry, and turn it into something that the rasterizer can understand
  - Input: arbitrary geometry, lighting and camera information
  - Output: Shaded screen-space polygons

Vertex Processing

- Vertex processing consists of:
  - ModelView transform
  - Projection transform
  - Lighting
  - Perspective divide
  - Polygon clipping
  - Viewport transform



Vertex Processing



Vertex Processing

- We already talked about lighting
- We already talked about modeling transformations
- We talked a bit about projections and viewports
  - We're going to do that in more detail today
- Clipping is for next time

Viewing in OpenGL

- The OpenGL viewpoint acts as a virtual camera
  - What parameters do you need to define a camera?
    - Viewpoint (Center of Projection)
    - View direction
    - Field of view
    - Film size
    - Projection plane

Viewing

- Viewing requires 3 elements:
  - Objects to be viewed
  - A viewer with a projection surface
  - A projection from the objects to the viewing surface

Viewing

- Example: A real camera
  - Objects: Whatever you're taking a picture of: landscape, people, etc.
  - Viewer: The camera (with its film as the projection surface)
  - Projection: Defined by the lens, maps 3D objects on to the 2D surface

Viewing

- Example: OpenGL camera
  - Objects: The input geometry
  - Viewer: The OpenGL "camera" (with the view volume as its viewing "surface")
  - Projection: The OpenGL projection matrix (GL_PROJECTION), maps 3D space (world coordinates) into 3D space (eye coordinates)
    - Eventually into normalized device coordinates (NDC)

Classical Viewing

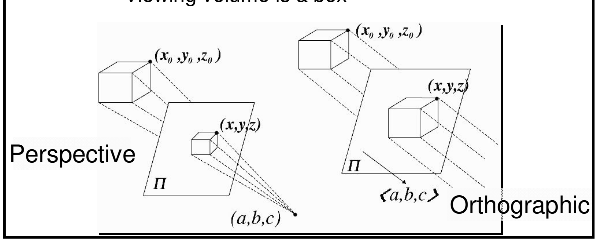
- Classical views are based on the relationship between these 3 elements: objects, a viewer, and a projection
- In classical views, objects are assumed to be constructed from flat principal faces
  - *i.e.* many buildings
- Used primarily by architects / engineers

Planar Geometric Projections

- Standard projections are assumed to be onto a single plane
- A projection can be perspective or orthographic
  - In perspective projection, all rays converge at a single point (the center of projection, or COP)
  - In orthographic projection, all rays are parallel

Perspective vs. Parallel

- Perspective
  - Viewing volume is a truncated pyramid
    - aka *frustum*
- Orthographic
  - Viewing volume is a box

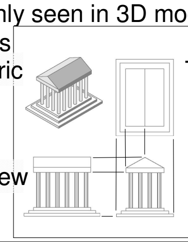


Taxonomy of Planar Geometric Projections

- Planar Geometric Projections
  - Parallel / Orthographic
    - Multiview
    - Axonometric
      - Isometric
      - Dimetric
      - Trimetric
    - Oblique
  - Perspective
    - One-point
    - Two-point
    - Three-point

Multiview Orthographic

- Projection plane parallel to principle face
- Multiview simply means generating an orthographic view for multiple faces
- Commonly seen in 3D modeling programs



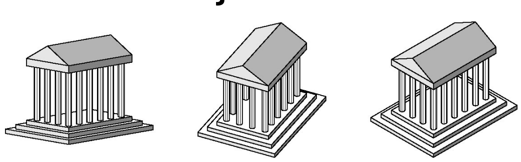
Benefits

- Why use orthographic?
  - Preserves distances and angles
    - Perspective does not
  - Can be used for measurements
- Why multiview?
  - The main problem with orthographic projection is that it hides many surfaces
  - Often add isometric view as well

Axonometric Projections

- Direction of projection is still perpendicular to the viewing plane
  - But principle faces not parallel to it
- 3 different kinds:
  - Isometric, dimetric, trimetric
  - Classified by number of different foreshortening factors

Axonometric Projections



Dimetric Trimetric Isometric

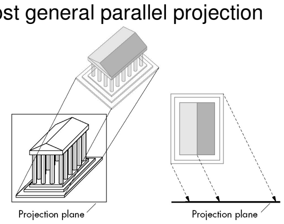
- Typically, one axis of space is drawn as the vertical (as seen here)

Advantages and Disadvantages

- Advantages:
  - Can see multiple faces of an object simultaneously
  - Lines are scaled, but by a constant factor
    - Could still be used for measurement
- Disadvantages:
  - Angles not preserved
  - Foreshortening does not depend on distance

Oblique Projection

- Direction of projection is not perpendicular to the viewing plane
  - Most general parallel projection



- Is this possible with a normal camera?

Orthogonal Projections

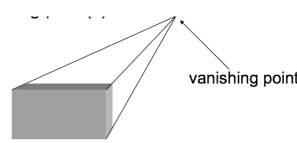
- So what is the matrix for an orthogonal projection?
  - Assume we're looking down z
- How would you implement an axonometric projection?
  - Orthogonal Projection + Rotation(s)
- How would you implement an oblique projection?
  - Orthogonal Projection + Rotation(s) + Shear(s)

Orthographic Examples

- How would you map an arbitrary bounding volume ($near_{xyz}$, far_{xyz}) into the volume defined by (-1, -1, -1) and (1, 1, 1)?

Vanishing Points

- In perspective projection, parallel lines (parallel in the scene) appear to converge to a single point
  - This is called the vanishing point

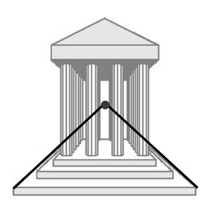


Perspective Projections

- Perspective projections are distinguished by the number of vanishing points in the image
  - One, two, or three

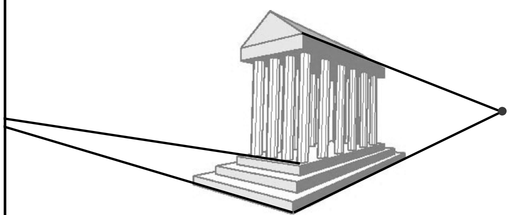
One-point Perspective

- One principal face is parallel to the projection plane



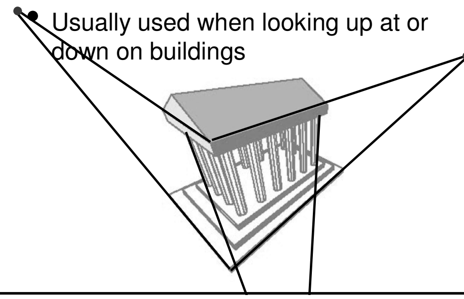
Two-point Perspective

- One principal direction (*i.e.* axis) is parallel to the projection plane



Three-point Perspective

- Nothing parallel to the projection plane
- Usually used when looking up at or down on buildings



Classical Viewing Recap

- Classical viewing is not “accurate”
  - Can be useful for various reasons
- Two main branches
  - Parallel projection
  - Perspective projection

Movie Break! The Aeronaut

Nicholas Lombardo
Ringling School of Art & Design SIGGRAPH 2006

Available online:
ttp://www.bestfilmoncampus.com/filmmaker/default.aspx?filmma

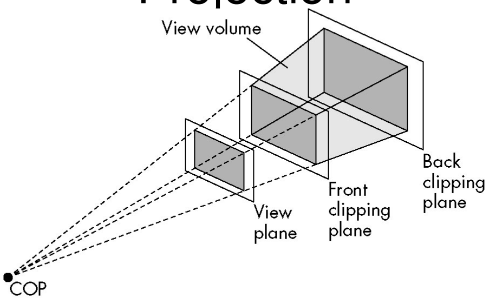
Doing Projections in OpenGL

- We already know the commands to set up projection matrices:
  - glOrtho(...)
  - gluOrtho2D(...)
  - glPerspective(...)
  - gluFrustum(...)
- Now we’ll talk a bit about what they really mean

Positioning the Camera

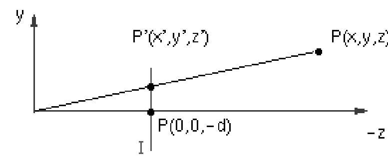
- We usually assume that the camera is located at (0,0,0), looking down -z
  - What do we do if we want it to appear somewhere else?
    - Translate the world by the opposite of the new location
  - Two ways to think about this:
    - Whole object moves into the camera frame
    - Camera moves (need to apply transforms in reverse order)

Perspective Projection



Perspective Divide

- Perspective divide is the mechanism by which objects farther away are made to appear smaller
  - How?



Perspective Divide

- x' = (x * d) / z
  y' = (y * d) / z
- How would you implement this in a matrix?
  - [1 0 0 0]
    [0 1 0 0]
    [0 0 1 0]
    [0 0 1/d 0]
    - d is the z location of the projection plane
- How does this work?

Perspective Divide

- M * $[x\ y\ z\ 1]^T = [x\ y\ z\ (z/d)]^T$
- Need to normalize new point:
  normalize($[x\ y\ z\ (z/d)]^T$) =
  [(xd/z) (yd/z) d 1]
- This is the image of that point on the projection plane
- What happens if d = 0?

Viewport Transform

- After perspective divide, we know where the vertices will map to in 2D
  - But in normalized device coordinates
- Need to know where these will actually be displayed
  - This is the viewport transform

Viewport Transform

- Just need to translate into a different set of 2D coordinates
  - From the rectangle defined by (-1, -1) and (1, 1) to the rectangle defined by (0, 0) and (width, height)
  - How?
    - Translate and scale
  - What if the aspect ratio of the viewport is different from that of the camera?

Next Time

- Continuing with vertex processing
  - Clipping
  - Discussion of vertex shaders
- Assignment 2 will go out
- Reminder: Programming assignment 1 due TONIGHT by 11:59pm
  - Upload to blackboard.unc.edu