course overview

www.csun.edu/~jb715473/math396.html
math for 3D graphics

- main idea: describe a scene in 3D space and render it to a 2D surface still looking 3D

- graphics hardware computes pixel data from vertex data ... of course doing this is not trivial

- why math?
  - geometry: project 3D object onto a plane
  - but there is more: how can the scene be described efficiently? how is the information about the color, the light, and the texture of an object stored?
  - and how does the hardware manipulate and display it?
the rendering pipeline - overview

- a 3D scene consists of separate **objects**, each defined by a set of **vertices** and a **graphics primitive**

- information about the object and its properties is stored at each vertex

- how that information is manipulated and displayed requires several transformations and a number of mathematical operations

- the sequence of transformations from vertices (input) to display (output) is known as the rendering pipeline

- in chapter 0, the book provides a detailed description of this process, which we'll summarize here
the rendering pipeline - hardware

- on modern computers graphics are handled by Graphical Processing Unit (GPU), separate from CPU
- CPU communicates with graphics application
- application sends rendering commands to graphics library (e.g., OpenGL)
- OpenGL sends commands to GPU (through a graphics driver)
- GPU processes vertex data to produce pixel data
- pixel data is displayed in image buffers
the rendering pipeline - hardware
the rendering pipeline - transforms

- several 3D coordinate systems some local, some global,
  - vertex data is stored in 3D **object** space - local
  - position of each object is given in **world** space - global
  - also **camera/eye** space: x and y align with display, and z in viewing direction

- **model-view transformation**

  object space  ➔  world space  ➔  camera space

  concatenate transforms: ➔  +  ➔
the rendering pipeline - transforms

- several 3D coordinate systems some local, some global,
  - vertex data is stored in 3D object space - local
  - position of each object is given in world space - global
  - also camera/eye space: x and y align with display, and z in viewing direction

- model-view transformation

object space ➔ camera space
the rendering pipeline - transforms

more transforms:

- projection: apply perspective

- performed in four-dimensional homogeneous clip space - graphics primitives clipped to visible area

- in clip space \(x, y, \text{ and } z\) are in \([-1,1]\), coordinates represent position vector of vertices

- viewport transformation of vertices from normalized coordinates to pixel coordinates - vertices now in window space
now some calculations:

- how much light reaches each vertex? how much is reflected?...

- **per-vertex lighting** - pixel light is interpolated from vertices

- vertices may also carry texture coordinates

- all these information is interpolated to determine the final color for each pixel in the viewport...
rasterization and fragment operations

once in viewport coordinates:

- **rasterization** - what pixels are covered by what primitive?
- **fragment** - depth, color, texture, and location of pixel
- **fragment/pixel shading** - fragment data is used to determine the final color and depth for each pixel
- several test are performed so as to determine what pixels will be visible before calculating fragment shading so as to avoid unnecessary operations
- if a pixel/fragment passes all tests, its color is **blended** to the buffer