Graphics Foundation

9/11/17

* Most video games are created using polygons (mainly triangles) for all textures, characters, structures, etc.
* Early games would use bitmaps to color in the pixels
  + Calculate where the pixels would be on the screen
  + Got more complicated as resolution got bigger, lighting got more advanced, and shading got more complex
* Software rendering
  + Ram->Video Card->Comp Screen
  + 2560\*1600 = 4,096,000 pixels => ~12mb
  + Sizeof(War and Peace)
  + 60 times a second
* Arcade games
  + Rendered Data onto Hardware skipping the video card step
  + 3D games before 3D on computer (more colorful and brighter)
* 1981/1982
  + First home console, very colorful graphics that ran quickly
  + Storing data on hardware
* Map and Distort Images to fit onto a 3D landscape.
  + Most efficient way of mapping images to polygons is triangles
* Late 1980s to 1990s
  + Gams were being programmed to be ran on the GPU allowing for more graphically intensive and complex
  + Just tinier triangles stored on the video card
* Hardware Rendering
  + Only copy the data needed
  + Ram->Video Card->Monitor
* No difference between 2D and 3D rendering
* GPU Pipeline
  + Two triangles, make up a square, in which the sprites are mapped
  + Vertex Data->Vertex Shader->Fragment Sader( also receiving textures)->Rasturize
  + Vertex Data
    - Defining the points that make up the triangle
    - A **polygon** is defined by points in space called **vertices**
    - Every point has a position on the cartesian plane
    - For every triangle, 6 numbers that describe the position of the vertices. (Focusing on 2D)
    - Order matter in how the triangle is described
      * Optimization = one side is invisible and doesn’t render the textures
    - Define them in counter-clockwise order
    - Screen = Area GPU is rendering
      * Measured in Normalized Device Coordinates
      * Aspect Ratios
  + Vertex Shader
    - A program that transforms the attributes (such as position, color or others) of every vertex passed to the GPU
  + Fragment Shader
    - A **program** that returns **the color of each pixel** when geometry rasterizes onto the **screen**.
  + Textures
    - Should be processed along with the fragment due to fragment is working with colors
  + Vectors
    - Direction 1-dimension: +1 or -1 on the x-axis
    - Direction 2-dimensions: 1,0,-1 x-axis; 1,0,-1 y axis in a matrix []
      * Changing numbers can either scale or rotate
  + Homogenous Coordinates
    - Take a unit system and take it up 1 dimension (XY1)
    - Used for skewing, transformations and projection mapping objects
  + Transformation matrices: Class Slide for images & examples