Midterm 1 Study Guide and Notes Sheet

* Transformations (scaling, translating, rotating)
  + Tcombined = TvpTprojTcameraTmodel
  + Viewport Transformation = first a translation then scale - ex: m4.scaling([200,-200,200]),m4.translation([200,200,0])
* Camera Coordinate System
  + xyz/uvw is the basis coordinate system
  + vector v = w × u, w = -g/||g||, u = t x w/||t x w||
  + “eye” vector (how you want to look at the object)
  + “gaze/target” vector (what you want the center of the camera to be looking at)
  + “up” vector (what you want to appear as vertical)
  + lookAt(eye, target, up) takes it from camera view to world view
    - therefore, must take inverse of lookAt to go from world to camera
* Homogenous Coordinates
  + allow common vector operations such as translation, rotation, scaling and perspective projection to be represented as a matrix by which the vector is multiplied
  + not only clean up the code for transformations, but this scheme also makes it obvious how to compose two affine transformations: simply multiply the matrices
  + cheaper to implement, as it eliminates a division operation
* Painter’s Algorithm vs. Z-Buffer
  + Painter’s Algorithm
    - needs to repeat the sorting of triangles if the camera position moves
    - can fail to produce the right result when some of the triangles to be drawn intersect one another
  + Z-Buffer
    - at each pixel we keep track of the distance to the closest surface that has been drawn so far, and we throw away fragments that are farther away than that distance, an image-based method applied during the rasterization stage, easy to be implemented on hardware, unlimited scene complexity, no need to sort the objects, no need to calculate object-object intersections, waste time drawing hidden objects, z-precision errors (aliasing problem)
* Rasterization
  + For each primitive that comes in, the rasterizer has two jobs: it *enumerates* the pixels that are covered by the primitive and it *interpolates* values, called attributes, across the primitives, a problem is aliasing and discretization which is that pixels are an integer grid
* Orthographic vs. Perspective Projections
  + Orthographic Projections
    - Parallel lines, height of objects, length of lines, angles between lines all stay the same
    - Uses parallel lines to project image onto a plane (projection lines are orthogonal to the projection plane)
    - gives us a very clear method of communicating ideas and objects
  + Perspective Projections
    - Size varies inversely with distance (making it look realistic), can't judge distances as exactly as we can with parallel projections
* Barycentric Coordinates
  + 0 <= alpha,beta,gamma <= 1, alpha + beta + gamma = 1
* Vertex Shader vs. Fragment Shader
  + Vertex Processor - the programmable stage in the rendering pipeline that handles the processing of individual vertices
  + Fragment Processor - a collection of operations applied to fragments generated by the rasterization operation in the rendering pipeline

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Examples** | **VP Accessibility** | **FP Accessibility** | **HP Accessibility** |
| **Attribute -** a specification that defines a property of an [object](https://en.wikipedia.org/wiki/Object_(computer_science)), element, or file | color, position, density, thickness | Yes | No | Yes |
| **Uniform –**  variables that have same value at every vertex and at every pixel of a primitive | camera view, lighting, time | Yes | Yes | Yes |
| **Varyings-**  provides an interface between Vertex and Fragment Shader, if you define a varying variable in a vertex shader, its value will be interpolated (over the primitive being rendered and you can access the interpolated value in the fragment shader, varyings can be used as write-only in the vertex shader and read-only in the fragment shader, and affect individual pixels whereas attributes affect a whole vertex | the color attribute was used to transform the vertex position, this time we actually use the color value in its correct purpose and pass it down (as varying variable) to the fragment shader | Yes (output) | Yes (input) | No |
| **Constants -**  a value that cannot be altered by the program during normal execution | PI | Yes | Yes | No |

* + - Fragment shader can access uniform variables, which are set by the host program, and also receive information indirectly via the vertex shader