Engine

1. core
   1. Display – Configuration and initialization of window elements
      1. Display – initial configuration variables
      2. getWindowId – returns windowId
      3. getTitle – returns title
      4. setTitle – passes title to GLFW
      5. getX – returns x (location in screen space)
      6. setX – passes x to GLFW
      7. getY – returns y (location in screen space)
      8. setPos – passes x and y into GLFW
      9. getW – returns w (window width)
      10. setW – passes w into GLFW
      11. getH returns h (window height)
      12. setH – passes h into GLFW
      13. setSize – passes w and h into GLFW
      14. getFullscreen – returns fullscreen state
      15. setFullscreen – passes a state into GLFW
      16. getVsync – returns vSync state
      17. setvSync – passes a vSync state into GLFW
      18. setVisible – passes a state into GLFW to change window visibility
      19. isClosed – checks window close state and closes
      20. create – uses glfwInit() to initialize window, sets important configuration
      21. update –to call in update loop, important glfw commands
      22. exit – destroys window
   2. Engine – contains main game loops, starts and stops engine
      1. Engine – initialization
      2. getDisplay – returns display instance
      3. getSceneHandler – returns sceneHandler instance
      4. getInputHandler – returns inputHandler instance
      5. start – calls Display.create, initializeses inputHandler resourceLoader and sceneHandler
      6. exit – exits sceneHandler, resourceLoader, display
      7. stop – used to terminate update loop
      8. updateLoop – master update loop for engine, defines engine time fps and tps, updates sceneHandler passes deltatime
         1. note update loop runs on a separate thread than the renderloop
      9. renderLoop – master render loop for engine, renders sceneHandler scene, updates display
      10. sleep – functionality for the engine time, allows the thread to sleep
   3. MasterRenderer – place to handle all rendering tasks
      1. MasterRenderer – initialization
      2. loadMatrix – loads mvp matrix into master renderer
      3. renderModel – for an untextured model, runs shader, loads matrix, renders model
      4. renderModel – for textured model (method overloading), identical but renders textured model
   4. ResourceLoader – place to load resources like textures and models
      1. ResourceLoader – initialization
      2. loadTexture – calls textureLoader, loads texture file
      3. exit – exits textureLoader
   5. Settings – underused but should be central place for all engine settings
      1. Settings – initialization
      2. setMaxTPS – set ticks per second
2. Input
   1. InputHandler – handles human input
      1. InputHandler – initialization
      2. isKeyDown – searches keyboard array for key
      3. isButtonDown – searches mousebutton array for button
      4. getXcursor – x position of cursor
      5. getYcursor – y position of cursor
      6. getXoffset – x offset of scrollwheel
      7. getYoffset – y offset of scrollwheel (more useful)
      8. registerInputHandler – uses GLFW callbacks to get input information
3. math
   1. Mat4f – creates a matrix datatype, handles all matrix related manipulation and construction, defines a 4x4 matrix of floats
      1. Mat4f – empty constructor
      2. Mat4f – constructor for float array
      3. Mat4f – constructor for 4 dimensional vector by columns (opengl is column major)
      4. Mat4f – constructor for 3 dimensional vector, w is set to 0
      5. m00() through m33() in m(row)(column) format getters
      6. m00() through m33() in m(row)(column) format setters
      7. set – sets existing matrix using new array of floats
      8. identity – converts matrix into identity matrix
      9. empty – converts matrix into empty matrix (all 0)
      10. projection – creates projection matrix using fov, aspectratio, znear ,and zfar
      11. translation – creates a translation matrix with x y and z
      12. string – used for debugging, deconstructs matrix into array of floats
      13. dot – performs dot product where the parameter is the right matrix and the instance is the left matrix, takes in a matrix
      14. get – uses put to convert matrix into float buffer
      15. put – takes a matrix and a float buffer and assigns all elements of the matrix to floatbuffer
   2. Vec3f – creates a 3 dimensional vector of floats datatype
      1. Vec3f – constructor
      2. getters and setters for x y and z
   3. Vec4f – creates a 4 dimensional vector of floats datatype
      1. Vec4f – constructor
      2. Vec4f – constructor from Vec3f to Vec4f with w = 1
      3. Vec4f – constructor from Vec3d to Vec4f with w parameter
      4. getters and setters for x y z and w
4. model
   1. Model – creates a model datatype that stores a vaoId and vertex count
      1. Model – initialization
      2. getVaoId – returns vaoId
      3. getVertexCount – returns model vertex count
      4. bind – binds vertex array object (vao) using opengl
      5. unbind – binds vertex array object to 0, emptying opengl binding
   2. ModelBuilder – main process to load model into opengl, heavy with opengl commands
      1. BuildModel – initializes vaoId, binds vao and indices, initializes vao buffers, returns model
      2. BuildModel – method overloading, identical but includes uvCoordinates and textures
      3. bindIndices – creates and uses a vbo (vertex buffer object) to load indices into gpu, Separate from storeDataInAttributeList because among other things indices must be loaded using GL\_ELEMENT\_ARRAY\_BUFFER vbos while storeDataInAttributeList uses GL\_ARRAY\_BUFFER
      4. storeDataInAttributeList – creates and uses a vbo to load in other vertex attributes such as color, normals, uv coordinates, etc.
      5. createVao – creates a new vao id
      6. createVbo – creates a new vbo id
      7. exit – cleanup, iterates through stored vao and vbo lists and deletes buffers
   3. ModelRenderer – master renderer for a model, calls methods in their necessary order
      1. ModelRenderer – constructor
      2. render – takes a model, binds it, renders it, unbinds it
      3. render – same except for textured model
      4. bindModel – calls model.bind() and enables vao buffers
      5. renderModel – uses glDrawElements to draw triangles
      6. unbindModel – disables vao buffers, calls model.unbind()
      7. bindTexturedModel – similar but for textured models
      8. renderTexturedModel – similar but for textured models
      9. unbindTexturedModel – similar but for textured models
   4. TexturedModel – extends model, adds texture information to datatype
      1. TexturedModel – initialization
      2. getTexture – returns texture object
      3. setTexture – sets texture object
5. scenes
   1. DefaultSceneIds – enum to define scene types, basically creates a datatype of sceneIds which can be used instead of number identifiers
   2. Scene – abstract constructor for scene object, a new scene will extend the scene class
      1. setShader – sets the shader object
      2. getShader – returns shader
      3. Scene – sets scene id
      4. sceneShading – creates a projection matrix with fov aspectratio znear and zfar
      5. initialize – abstract method, takes in ResourceLoader
      6. update – abstract method, takes in deltaTime, InputHandler
      7. render – abstract method, takes in MasterRenderer
      8. enter – currently unused, things to do on scene entry
      9. leave - currently unused, things to do on scene leaving
      10. exit - currently unused, things to do on exiting engine
      11. getSceneId – returns sceneId
   3. SceneHandler – handles scene loading, registering, storing, organizing etc.
      1. SceneHandler – initializes scene hashmap, sets scene to splashscreen (default loading screen)
      2. registerScene – adds scene to hashmap
      3. removeScene – removes scene from hashmap
      4. setScene – sets active scene, leaves old scene, enters new scene
      5. initializeRegisteredScenes – iterates through registered scenes and calls initialization method and passes in ResourceLoader
      6. updateActiveScene – brings active scene into the update loop, calls active scene update
      7. renderActiveScene – brings active scene into render loop, calls active scene render
      8. exit – leaves active scene, iterates over registered scenes and exits
   4. SplashScreen – extends scene, default scene to display when no scene is loaded
      1. initialize – currently empty
      2. update – currently empty
      3. render – currently empty
6. shaders
   1. Shader
   2. StaticShader
7. test
   1. TestMain
   2. TestScene
8. textures
   1. Texture
   2. TextureLoader