SD4 Assignment 3: 3D Object Viewer & Cooker

# Summary

* Creation of a new project (cloned from Protogame3D) named “ModelViewer” or something similar
* Ability to load and display single-mesh, single-material 3D .OBJ static models (one at a time)
* Ability to save, load, and display single-mesh, single-material 3D .GHSM (or custom binary format) static models
* Conformance to the [.GHSM format specification](https://docs.google.com/document/d/1Tx9pJX_ub1gUyy5zlKqCS-ye0sHSqlevLC41F6tF7to/edit) OR your own file format specification for a custom file format
* Ability to automate “cooking” of .OBJ to .GHSM (or custom binary) files using run-and-quit “commandlets”
* “Load” & “Save” dev console commands for loading (of .OBJ or .GHSM/custom) & saving (.GHSM/custom) models
* Ability to render the currently-loaded 3D model using a simplified Blinn-Phong (“bumpy shiny glowy”) shader
* Ability to fly around the model using 5-DoF camera controls
* Ability to toggle/cycle a variety of rendering options for the currently-loaded 3D model

# Requirements (maximum possible score is 100/100):

1. **(10 points) .OBJ model loading**
   1. Ability to load most basic .OBJ 3D static model formats to a CPU mesh (vertex array)
      * Note: I need objs to have vt’s defined in order to properly calculate tangents for shading calculations. Also, negative face indices and a few of the sample objs gave me trouble. I can texture/light Woman, Bunny, CrystalBall, Ship, and Box for sure
   2. May use vertex indexing in CPU memory or not (your choice)
   3. Ability to scale or axis-swap an .OBJ at load time based on meta-data #comments in the .OBJ itself
   4. Need not support: .OBJ: negative indexing, smoothing groups (‘s’), material (.mtl) references, etc.
   5. Can send mesh to GPU and render as VBO (indexed or not)
2. **(25 points) .GHSM (or non-GHSM custom binary format) Loading and Saving**
   1. Ability to save a loaded CPU mesh (vertex array) as a proper .GHSM (or custom binary) model file
   2. Ability to load a .GHSM (or custom binary) model file into memory (as CPU mesh / vertex array) from disk
   3. May use vertex indexing or not (your choice)
   4. Mesh data loaded must be *ready to render as-is*, i.e. just load data to a buffer, send buffer to GPU, draw.
3. **(15 points) .OBJ-to-binary (.GHSM or custom) “cooking” dev console commands and commandlet**
   1. Ability to load a static mesh (.OBJ or .GHSM/custom) file via a “Load” or similar dev console command
   2. Ability to save a static mesh (.GHSM/custom) file via a “Save” or similar dev console command
   3. Ability to launch the project with command line arguments that cause it to run as a “commandlet”;
      * Commandlet is “executed” and then App shuts down immediately – no main loop entered.
      * No engine subsystems (e.g. Window, Renderer) are created unless the Commandlet creates (and destroys) them manually.
   4. At least one commandlet, which “cooks” a named .OBJ (loads it, and saves it) to a .GHSM/custom file.
      * Note: Uses Data/ as root path, so you’ll need to type Models/Bunny to cook Bunny.obj into Bunny.twsm
4. **(10 points) Camera & controls**
   1. WASD (or ESDF) controls for horizontal movement (forwardXY, leftXY, backXY, rightXY)
      * Note: WASD flys around in 3 dimensions, but EQ still moves vertically so a similar affect can be achieved
   2. EQ (or AZ or RW or QZ or EX or similar) controls for vertical movement (skywardZ, downwardZ)
   3. MouseX controls Yaw (about your Up axis), unbounded wraparound
   4. MouseY controls Pitch (about your left/right axis), clamped to [-89.9,+89.9]
   5. Keyboard flying speeds AND mouse movement speeds should be *reasonable* and *framerate-independent*
5. **(25 points) Simplified Blinn-Phong render pipeline**
   * + Note: The pipeline is implemented but I did notice some black pixel artifacting around edges of the cobblestones texture when specular is on, so I may have some small error in my specular calculation assuming it’s not caused by the resolution of the texture.
   1. Vertexes have (at least) the following attributes: Position, Color, UV, Tangent, Bitangent\*, Normal
      * *\*Bitangent may be omitted & reconstructed in shader if its handedness is stored in Tangent.w*
      * T,B,N vectors are ortho-normalized in the pixel shader and used to form a TBN (tangent-to-world) transformation matrix
   2. Diffuse map is bound and sampled and rendered at the correct UV positions
      * If no Diffuse map is provided, an internal 1x1 white texture is bound instead
   3. Normal map (tangent space / light blue-ish) is bound and sampled, decoded, and renormalized
      * Decoded normal map normals are transformed into world space using a TBN transform matrix
        + *(or, alternatively, all camera/lights are transformed into tangent space using TBN-1)*
      * If no Normal map is provided, an internal 1x1 light blue (.5,.5,1) texture is bound instead
   4. EITHER:
      * A combined Specular/Glossy/Emissive map is bound and sampled; the R, G, and B channels are each interpreted as greyscale Specular Map, Glossiness Map, and Emissivity Map, respectively:
        + Specularity [0,1] is taken directly from the greyscale Specular Map (Red channel)
        + Glossiness [0,1] is taken directly from the greyscale Glossiness Map (Green channel)
        + Emissivity [0,1] is taken directly from the greyscale Emissivity Map (Blue channel)
      * Specular color is calculated as: Specularity \* Light Color \* Light Intensity
      * Specular exponent is calculated as: a range-map of Glossiness in [0,1] into [1,32]
      * Emissive color is calculated as: Diffuse.rgb \* Emissivity
      * If no Spec/Gloss/Emit map is provided, a 1x1 dark yellow (.5,.5,0) texture is bound instead
   5. OR:
      * Separate Specular, Gloss, and Emissive map textures are each separately bound and sampled:
        + Specular color is Specular texel.rgb \* Light Color \* Light Intensity
        + Glossiness [0,1] is taken from the Red channel of the Gloss map texel (G,B ignored)
        + Emissive color is taken directly from the Emissive map texel.rgb
      * If no Specular map is provided, an internal 1x1 grey (.5,.5,.5) texture is bound instead
      * If no Glossy map is provided, an internal 1x1 grey (.5,.5,.5) texture is bound instead
      * If no Emissive map is provided, an internal 1x1 black (0,0,0) texture is bound instead
   6. Specular highlight color is added to the final pixel output color, and is computed roughly as:
      * pow( saturate( dot( reflectedCamToPixelDir, pixelToLightDir ) ), specularExponent )
   7. Emissive color is added to the final pixel output color
   8. If multiple lights are supported, diffuse light is summed for all lights (and multiplied by several factors) then applied to (multiplied vs.) the diffuse texture & surface, with falloff; specular highlight color is separately computed per light and summed together; emissive color and total specular are added last.
6. **(15 points) Toggleable/cycleable render features**
   1. Your model viewer must be able to toggle on/off (or cycle through) each of the following render options:
      * Fullbright (disable lighting/shading calculations altogether – maybe bind a different shader?)
      * Use of Diffuse map (if disabled, bind internal white texture instead)
      * Use of Normal map (if disabled, bind internal light-blue texture instead)
      * Use of Spec/Gloss/Emissive map(s) (if disabled, bind appropriate default replacements instead)
      * Diffuse lighting (calculation results in a black diffuse object, may have specular and/or emissive)
      * Specular highlighting
      * Emissive glow
      * Wireframe (toggleable, OR can render front faces (outside) solid, back faces (inside) wireframe)
      * Various lighting modes (e.g. on-camera, off-camera, orbiting, etc.) – at least three
   2. All controls and keyboard shortcuts must be clearly documented onscreen and/or in a ReadMe file.

# Submission

* Use your own Guildhall “SD” C++ engine – maintained, buddy-tested, and submitted via Perforce.
  + A committed changelist in P4 with submission comment “SD4-A3: COMPLETE”.
  + Be sure to include an updated ReadMe, a current Release-built .EXE, and all required code & data files.
  + *See notes from Professor Service’s DFS2 class regarding maintenance of your code across P4 / GitHub.*
* Submit a **.zip file** to Canvas named **C29\_SD4\_*A3*\_*p4username*.zip** (*for example:* ***C29\_SD4\_A3\_beiserloh.zip***) which contains:
  + A very short (informal) voice-narrated **video** demonstrating the full functionality of your project;
  + A copy of **this document**, with completed items highlighted cyan, omitted items highlighted red, and partially completed items highlighted yellow (with inserted bullets-text underneath explaining).