You need:

[] – 3D assets, in either OBJ or GLTF form

* TheStanleyParable.obj – 1 instance - MTSP
* Clock.obj – 1 instance – MClock
* Computer.obj – 1 instances – MComputer
* Lamp.obj – 1 instances – MLamp
* Painting.obj – 1 instances – MPainting
* PaperTray.obj – 3 instances – MPaperTray
* Sharpener.obj – 1 instances – MSharpener

[] – 3D assets dynamically generated in the code

* Todo – Todo – Todo – 1 instance - MTodo

[] – Textures associated with the models

* TheStanleyParable.png – TTSP
* Clock.png – TClock
* Computer.png – TComputer
* Lamp.png – TSplash
* Painting.png - TPainting
* PaperTray.png – TpaperTray -- TODO: 3?
* Sharpener.png - TSharpener

Then you decide:

[] – the illumination for the scene:

[] – which type of direct light? How many ?

1 – Spot Light from Lamp + Direct Lightining from Ambient

[] – Ambient light type?

Constant Ambient Light

[] – Any object having emission?

1 – Computer

* These terms might be enclosed in a 3 scene-wide DescriptorSetLayout
  + gubo DescriptorSetLayout including:
    - Direct light color
    - Direct light position
    - Ambient light color
    - Viewer position
      * struct GlobalUniformBlock
  + DSLGubo
    - 1 UNIFORM block including the data above
* For each asset

MTSP, Mclock, Mcomputer, Mlamp, Mpainting, MpaperTray, MSharpener

* + [] – Define which vertex format it uses
    - Position
    - Normal vector
    - UV
      * Struct VertexMesh
  + [] – Select a BRDF approximation and shading technique, and depending on the scene illumination, define the corresponding Vertex / Fragment shader couple
    - Phong smooth shading
    - Lambert + Bilnn BRDF
  + [] – Decide which texture it requires
    - Color texture
  + [] – Decide which data sent from the CPP code the shaders need
    - Specular color
    - Specular power
    - Ambient sensitivity
    - ---------------------------------------------
    - World-view-projection matrix
    - World matrix
    - Normal transform matrix
      * struct MeshUniformBlock
    - The last two point determines the DescriptorSetLayout for the shader couple
      * 1 UNIFORM block including the data above
      * 1 Texture with the corresponding color
        + DSLMesh

You then:

[] – Examine how many different formats have been used by the assets

One -> see above

* Vmesh

[] – How many different DescriptorSetLayout are needed

Two -> see above

[] – How many different vertex and fragment shaders are needed

* This will also determine how many pipelines are needed
  + PMesh
    - Vertex Shader: MeshVert.spv
    - Fragment Shader: MeshFrag.spv
    - Based on VMesh and {DSLGubo, DSLMesh}

You can then:

[] – Create the Vertex formats

[] – Define the models and load them

[] – Define the texture and load them

[] – Create a DescriptorSetLayout for the scene-wide and pipeline specific uniform

[] – Create the pipelines needed

[] – For each scene-wide DescriptorSetLayout, create the corresponding DescriptorSet instance

* DSGubo – instances DSLGubo
  + struct GlobalUniformBlock
* DSTSP, DSclock, DScomputer, DSlamp, DSpainting, DSpaperTray, DSSharpener
  + struct MeshUniformBlock

[] – Count the required number of:

* DescriptorSets: 9
  + DSGubo, DSTSP, DSclock, DScomputer, DSlamp, DSpainting, DSpaperTray, DSSharpener
* UniformBlocks elements of the DescriptorSets:
  + All DS
* Texture elements of the DescriptorSets:
  + All DS except DSGubo

[] – For each 3D asset, create its specific DescriptorSet according to the corresponding DescriptorSetLayout. Here is where you will define the size of the corresponding uniform, and assign the textures.

* Init the variables above

[] – In the procedure that populates the command buffer, enter the command to draw all the primitives:

[] – first bind the scene-wide DescriptorSets

[] – for each different pipeline:

* + [] - Bind the pipeline
  + [] - For each object belonging to that pipeline:
    - [] – Bind the corresponding DescriptorSet
    - [] – Bind the vertex and index buffers
    - [] – call the draw command for the corresponding mesh
* Remember: it is always easier to load all the 3D objects at the beginning, and then “hide” the ones you do not need by either giving them a zero scale, or by moving them far away from the far plane of the camera.

[] – initialize all the variables for the game logic

* Here I initialize the DescriptorSets and map to set the initial state of the objects

[] – in the procedure that handles the user interaction:

[] – Read the user input (from the keyboard, the mouse or the Joystick)

* + Orbiting camera model – left stick moves camera forward or up / down, right thumb moves the camera around the slot machine. Implented by storing the target position and the camera position and using a LookAt matrix
    - Four float variables needed: CamH, CamRadius, CamPitch, CamYaw
  + Implement the state machine of the game

[] – update the camera position and direction (if needed), and the corresponding view / projection matrix

* + Camera FoV = 90 deg, near plane = 0.1, far plane = 100

[] – update the variable with the position of the objects

[] – determine the new values of the uniform variable and map them

**1 - Vertex formats (C++)**

|  |  |
| --- | --- |
| **Name** | **Data structure** |
| VertexMesh | struct VertexMesh {  glm::vec3 pos;  glm::vec3 norm;  glm::vec2 UV;  }; |
| VertexOverlay | struct VertexOverlay {  glm::vec2 pos;  glm::vec2 UV;  }; |

**2 - Data structures for Uniform Block Objects (C++)**

|  |  |
| --- | --- |
| **Name** | **Data structure** |
| GlobalUniformBlock | struct GlobalUniformBlock {  alignas(16) glm::vec3 DlightDir;  alignas(16) glm::vec3 DlightColor;  alignas(16) glm::vec3 AmbLightColor;  alignas(16) glm::vec3 eyePos;  }; |
| MeshUniformBlock | struct MeshUniformBlock {  alignas(4) float amb;  alignas(4) float gamma;  alignas(16) glm::vec3 sColor;  alignas(16) glm::mat4 mvpMat;  alignas(16) glm::mat4 mMat;  alignas(16) glm::mat4 nMat;  }; |

**3 - Descriptor Set Layouts**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Binding** | **Type** | **Which shader** |
| DSLMesh | 0 | UBO | ALL |
| 1 | Texture | Fragment |
|  |  |  |
| DSLGubo | 1 | UBO | ALL |
|  |  |  |
|  |  |  |

**4 - Vertex Descriptors**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Format (C++)** | **Location** | **Type** | **Usage** |
| VMesh | VertexMesh | 0 | vec3 | POSITION |
| 1 | vec3 | NORMAL |
| 2 | vec2 | UV |

**5 - Pipelines**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Variable** | **Vertex Shader** | **Fragment Shader** | **Vertex format (C++)** | **Vertex descriptor** | **Set ID** | **Descriptor set Layout** |
| PMesh | MeshVert.spv | MeshFrag.spv | VertexMesh | VMesh | 0 | DSLGubo |
| 1 | DSLMesh |
|  |  |
| PEmitter | EmitterVert.spv | EmitterFrag.spv | VertexMesh | VMesh | 0 | DSLGubo |
| 1 | DSLMesh |
|  |  |

**6 - Mesh objects**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Vertex Format (C++)** | **Vertex descriptor** | **Type** | **Model File** |
| MTSP | VertexMesh | VMesh | OBJ |  |
| MClock | VertexMesh | VMesh | OBJ |  |
| MComputer | VertexMesh | VMesh | OBJ |  |
| MLamp | VertexMesh | VMesh | OBJ |  |
| MPainting | VertexMesh | VMesh | OBJ |  |
| MPaperTray | VertexMesh | VMesh | OBJ |  |
| MSharpener | VertexMesh | VMesh | OBJ |  |

**7 - Textures**

|  |  |  |
| --- | --- | --- |
| **Variable** | **File** | **Sampler** |
| TTSP | TSP.png | - |
| TClock | Clock.png | - |
| TComputer | Computer.png | - |
| TLamp | Lamp.png | - |
| TPainting | Painting.png | - |
| TPaperTray | PaperTray.png | - |
| TSharpener | Sharpener.png | - |

**8 - Uniform Blocks Objects, C++ sides**

|  |  |
| --- | --- |
| **Type** | **Variable** |
| MeshUniformBlock | uboTSP |
| MeshUniformBlock | uboClock |
| MeshUniformBlock | uboComputer |
| MeshUniformBlock | uboLamp |
| MeshUniformBlock | TuboPainting |
| GlobalUniformBlock | gubo |
| MeshUniformBlock | uboPaperTray |
| MeshUniformBlock | uboSharpener |

**9 - Descriptor Sets**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Variable** | **Descriptor Set Layout** | **Binding** | **Type** | **C++ data structure** | **Variable with values** | **Texture** |
| DSBody | DSLMesh | 0 | UBO | MeshUniformBlock | uboBody |  |
| 1 | Texture |  |  | TBody |
|  |  |  |  |  |
| DSHandle | DSLMesh | 0 | UBO | MeshUniformBlock | uboHandle |  |
| 1 | Texture |  |  | THandle |
|  |  |  |  |  |
| DSWheel1 | DSLMesh | 0 | UBO | MeshUniformBlock | uboWheel1 |  |
| 1 | Texture |  |  | TWheel |
|  |  |  |  |  |
| DSWheel2 | DSLMesh | 0 | UBO | MeshUniformBlock | uboWheel2 |  |
| 1 | Texture |  |  | TWheel |
|  |  |  |  |  |
| DSWheel3 | DSLMesh | 0 | UBO | MeshUniformBlock | uboWheel3 |  |
| 1 | Texture |  |  | TWheel |
|  |  |  |  |  |
| DSRoom | DSLVColor | 0 | UBO | MeshUniformBlock | uboRoom |  |
|  |  |  |  |  |
|  |  |  |  |  |
| DSKey | DSLOverlay | 0 | UBO | OverlayUniformBlock | uboKey |  |
| 1 | Texture |  |  | TKey |
|  |  |  |  |  |
| DSSplash | DSLOverlay | 0 | UBO | OverlayUniformBlock | uboSplash |  |
| 1 | Texture |  |  | TSplash |
|  |  |  |  |  |
| DSGubo | DSLGubo | 0 | UBO | GlobalUniformBlock | gubo |  |
|  |  |  |  |  |
|  |  |  |  |  |

**10 - Scene Objects**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Pipeline** | **Mesh** | **Descriptor Sets** |
| Room | PMesh | MRoom | DSGubo |
| DSRoom |
| Clock | PMesh | MClock | DSGubo, |
| DSClock |
| Computer | PMesh | MComputer | DSGubo |
| DSComputer |
| Lamp | PMesh | MLamp | DSGubo |
| DSLamp |
| Painting | PMesh | MPainting | DSGubo |
| DSPainting |
| PaperTrail | PMesh | MPaperTrail | DSGubo |
| DSPaperTraik |
| Sharpener | PMesh | MSharpener | DSGubo |
| DSSharpener |