You need:

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

* Tile.obj – 144 instances – MTile

[] – 3D assets dynamically generated in the code

* Background – quad – world coordinates 3D – 1 instance - MBackground

[] – Textures associated with the models

* background/poolcloth.png – TpoolCloth
* tiles/white.png - TwhiteTiles

Then you decide:

[] – the illumination for the scene:

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

1 – direct light from above (decide on type: point/sun/spotlight)

[] – Ambient light type?

Constant ambient

[] – Any object having emission?

No (double check for selected tiles and darkness in scene)

* These terms might be enclosed in a 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

MTile

* + [] – 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 + Blinn BRDF
  + [] – Decide which texture it requires
    - RGBA texture 🡪 TwhiteTiles
  + [] – 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 (might be unnecessary)
    - ---------------------------------------------
    - TileIndex (int)
      * struct TileUniformBlock
    - The last two point determines the DescriptorSetLayout for the shader couple
      * 1 UNIFORM block including the data above
      * 1 Texture with the corresponding color
        + DSLTile

MBackground

* + [] – Define which vertex format it uses
    - Position (3D world coordinates)
    - Normal
    - UV
      * Struct VertexMesh
  + [] – Select a BRDF approximation and shading technique, and depending on the scene illumination, define the corresponding Vertex / Fragment shader couple
    - Lambert + Oren Nayar
  + [] – Decide which texture it requires
    - RGBA Texture 🡪 TpoolCloth
  + [] – Decide which data sent from the CPP code the shaders need
    - Specular color
    - Specular power
    - Ambient sensitivity
    - Other Oren Nayar parameters
      * struct BackgroundUniformBlock
    - The last two point determines the DescriptorSetLayout for the shader couple
      * 1 Texture with the corresponding color
      * 1 UNIFORM block including the data above
        + DSLBackground

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

* DSLGubo
* DSLTile
* DSLBackground

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

* This will also determine how many pipelines are needed
  + PTile
    - Vertex Shader: PhongVert.spv
    - Fragment Shader: TileFrag.spv
    - Based on VMesh and {DSLGubo, DSLTile}
  + PBackground
    - Fragment Shader: LambertON.spv
    - Based on VMesh and {DSLGubo, DSLBackground}

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
* DSTile[144]– 144 instances of DSLTile
  + struct TileUniformBlock
* DSBackground – instance DSLBackground
  + struct BackgroundUniformBlock

[] – Count the required number of:

* DescriptorSets: 146
  + DSGubo, DSTile\*144, DSBackground
* UniformBlocks elements of the DescriptorSets: 146
  + All DS
* Texture elements of the DescriptorSets: 145
  + 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
  + Mouse input to select tiles
  + 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 🡪 decide on output

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

* + Disappearing tiles

[] – 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;  }; |

**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;  }; |
| TileUniformBlock | struct TileUniformBlock {  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;  alignas(4) int tIdx;  }; |
| BackgroundUniformBlock | struct BackgroundUniformBlock {  alignas(4) float amb;  alignas(4) float gamma;  alignas(16) glm::vec3 sColor;  alignas(?) ? Oren-Nayar params  }; |

**3 - Descriptor Set Layouts**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Binding** | **Type** | **Which shader** |
| DSLTile | 0 | UBO | ALL |
| 1 | Texture | Fragment |
|  |  |  |
| DSLBackground | 0 | UBO | ALL |
| 1 | Texture | Fragment |
|  |  |  |
| DSLGubo | 0 | 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** |
| PTile | PhongVert.spv | TileFrag.spv | VertexMesh | VMesh | 0 | DSLGubo |
| 1 | DSLTile |
|  |  |
| PBackground | - | LambertON.spv | VertexMesh | VMesh | 0 | DSLGubo |
| 1 | DSLBackground |
|  |  |

**6 - Mesh objects**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Vertex Format (C++)** | **Vertex descriptor** | **Type** | **Model File** |
| MTile | VertexMesh | VMesh | OBJ | Tile.obj |
| MBackground | VertexMesh | VMesh | Manual | - |

**7 - Textures**

|  |  |  |
| --- | --- | --- |
| **Variable** | **File** | **Sampler** |
| TpoolCloth | background/poolcloth.png | - |
| TwhiteTiles | tiles/white.png | - |

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

|  |  |
| --- | --- |
| **Type** | **Variable** |
| GlobalUniformBlock | gubo |
| TileUniformBlock | Tileubo\*144 |
| BackgroundUniformBlock | bgubo |

**9 - Descriptor Sets**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Variable** | **Descriptor Set Layout** | **Binding** | **Type** | **C++ data structure** | **Variable with values** | **Texture** |
| DSGubo | DSLGubo | 0 | UBO | GlobalUniformBlock | gubo |  |
|  |  |  |  |  |
|  |  |  |  |  |
| DSTile\*144 | DSLTile | 0 | UBO | TileUniformBlock | tileubo |  |
| 1 | Texture |  |  | Ttile |
|  |  |  |  |  |
| DSBackground | DSLBackground | 0 | UBO | BGUniformBlock | bgubo |  |
| 1 | Texture |  |  | TpoolCloth |
|  |  |  |  |  |

**10 - Scene Objects**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Pipeline** | **Mesh** | **Descriptor Sets** |
| Tile \* 144 | PMesh | MTile | DSGubo |
| DSTile |
| Background | PMesh | MBackground | DSGubo, |
| DSBackground |