You need:

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

- SlotBody.obj – 1 instance - MBody
- SlotHandle.obj – 1 instance - MHandle
- SlotWheel.obj – 3 instances – MWheel
- Room.obj - MRoom

[] – 3D assets dynamically generated in the code

- Splash screen – quad – only normalized screen coordinates 2D – 1 instance - MSplash
- Key press – quad – only normalized screen coordinates 2D – 1 instance - MKey

[] – Textures associated with the models

- SlotBody.png – Tbody
- SlotHandle.png – Thandle
- SlotWheel.png – Twheel
- SplashScreen.png – Tsplash
- PressSpace.png - TKey

Then you decide:

[] – the illumination for the scene:

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

1 – direct light from the back

[] – Ambient light type?

Constant ambient

[] – Any object having emission?

No

- 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

MBody, MHandle, MWheel1, MWheel2, MWheel3

  - [] – 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

MSpalsh, MKey

  - [] – Define which vertex format it uses
    - Position (2D normalized screen coordinates)
    - UV
      - Struct VertexOverlay
  - [] – Select a BRDF approximation and shading technique, and depending on the scene illumination, define the corresponding Vertex / Fragment shader couple
    - No illumination, just pass the UV and return the pixel read from the texture
  - [] – Decide which texture it requires
    - Color Texture
  - [] – Decide which data sent from the CPP code the shaders need
    - Visibility
      - OverlayUniformBlock
    - The last two point determines the DescriptorSetLayout for the shader couple
      - 1 Texture with the corresponding color
      - 1 UNIFORM block including the data above
        - DSLOverlay

MRoom

- [] – Define which vertex format it uses
  - Position
  - Normal vector
  - Color
    - Struct VertexVColor
- [] – 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
  - No 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
      - DSLVColor

You then:

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

Three -> see above

- VMesh
- VOverlay
- VVColor

[] – How many different DescriptorSetLayout are needed

Four -> 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}
    - POverlay
        - Vertex Shader: OverlayVert.spv
        - Fragment Shader: OverlayFrag.spv
        - Based on VOverlay and {DSLGubo, DSLOverlay}
    - PVColor
        - Vertex Shader: VColorVert.spv
        - Fragment Shader: VColorFrag.spv
        - Based on VVColor and {DSLGubo, DSLVColor}

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
- DSBody, DSHandle, DSWheel1, DSWheel2, DSWheel3 – 5 instances of DSLMesh
    - struct MeshUniformBlock
- DSRoom
    - struct MeshUniformBlock
- DSSplash, DSKey – instance DSLOverlay
    - struct OverlayUniformBlcok

[] – Count the required number of:

- DescriptorSets: 9
    - DSGubo, DSBody, DSHandle, DSWheel1, DSWheel2, DSWheel3, DSSplash, DSKey, DSRoom
- UniformBlocks elements of the DescriptorSets: 9
    - All DS
- Texture elements of the DescriptorSets: 7
    - 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 = 45 deg, near plane = 0.1, far plane = 100

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

- Only rotations for the wheels and the handle are needed:
  - Four float variables: HandleRot, Wheel1Rot, Wheel2Rot, Wheel3Rot

[] – 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;
	};

VertexVColor	struct VertexVColor {
	glm::vec3 pos;
	glm::vec3 norm;
	glm::vec3 color;
	};

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;
	};

OverlayUniformBlock	struct OverlayUniformBlock {
	alignas(4) float visible;
	};

Note: the new type of object uses the same data as the one in the *MeshUniformBlock*. For this reason, it is not necessary to add a new data structure for the C++ side of the uniform, and the same defined for the other 3D objects can be reused.

3 - Descriptor Set Layouts

Variable	Binding	Type	Which shader
DSLMesh	0	UBO	ALL
	1	Texture	Fragment
DSLOverlay	0	UBO	ALL
	1	Texture	Fragment
DSLVColor	0	UBO	ALL
DSLGubo	1	UBO	ALL

4 - Vertex Descriptors

Variable	Format (C++)	Location	Type	Usage
VMesh	VertexMesh	0	vec3	POSITION
		1	vec3	NORMAL
		2	vec2	UV
VVColor	VertexVColor	0	vec3	POSITION
		1	vec3	NORMAL
		2	vec3	COLOR
VOverlay	VertexOverlay	0	vec2	OTHER
		1	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
PVColor	VColorVert.spv	VColorFrag.spv	VertexVColor	VVColor	0	DSLGubo
					1	DSLVColor
POverlay	OverlayVert.spv	OverlayFrag.spv	VertexOverlay	VOverlay	0	DSLOverlay

6 - Mesh objects

Variable	Vertex Format (C++)	Vertex descriptor	Type	Model File
MBody	VertexMesh	VMesh	OBJ	SlotBody.obj
MHandle	VertexMesh	VMesh	OBJ	SlotHandle.obj
MWheel	VertexMesh	VMesh	OBJ	SlotWheel.obj
MKey	VertexOverlay	VOverlay	Manual	-
MSpalsh	VertexOverlay	VOverlay	Manual	-
MRoom	VertexVColor	VVColor	OBJ	Room.obj

7 - Textures

Variable	File	Sampler
TBody	SlotBody.png	-
THandle	SlotHandle.png	-
TWhell	SlotWheel.png	-
TKey	PressSpace.png	-
TSplash	SplashScreen.png	-

Note: the new object stores the color in the vertices. For this reason, no new texture is required.

8 - Uniform Blocks Objects, C++ sides

Type	Variable
MeshUniformBlock	uboBody
MeshUniformBlock	uboHandle
MeshUniformBlock	uboWheel1
MeshUniformBlock	uboWheel2
MeshUniformBlock	uboWheel3
GlobalUniformBlock	gubo
OverlayUniformBlock	uboKey
OverlayUniformBlock	uboSplash
MeshUniformBlock	uboRoom

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
Slot Body	PMesh	MBody	DSGubo
			DSBody
Slot Handle	PMesh	MHandle	DSGubo,
			DSHandle
Slot Wheel 1	PMesh	MWheel	DSGubo,
			DSWheel1
Slot Wheel 2	PMesh	MWheel	DSGubo
			DSWheel2
Slot Wheel 3	PMesh	MWheel	DSGubo
			DSWheel3
Room environment	PVColor	MRoom	DSGubo,
			DSRoom
Splash Screen	POverlay	MSplash	DSSpalsh
Press a Key sign	POverlay	MKey	DSKey