View

This module serves as a wrapper to all rendering and animation related engines. It’s the only class of the graphics side that is accessed by other modules. More specifically it should be accessed by the State System and the Object Manager. The State System will use it to initialize the graphic engines, update and render everything, while the Object Manager will use it to add a render instance when it’s creating a new object. All its functions are static, so there’s no need to keep an instance of this class anywhere. The View will contain private static instances of some graphic engines, and the other rendering-related managers will be friends of the View so that they can access those objects whenever they need to.

**Dependencies**

* Access to the following:
  + Renderer
  + Shader Manager
  + Texture Manager
  + Mesh Manager
  + Effect Manager
  + HUD Manager
  + Animation Manager
  + Render Instance
* Accessed by the following:
  + State System
  + Object Manager
  + All other major rendering managers are friends of this class and basically access it to get to any other rendering manager they need to get to.

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Bool | Initialize | HWND hWnd  // handle to the window we’re rendering to | Initializes all the render engines based on the window parameters. Returns true if succeeded, false otherwise. |
| Void | SetPlayer | Player\* pPlayer  // pointer to the player object | Sets the player pointer for the HUD class and the Camera. |
| Int | AddRenderInstance | BaseObject\* pObject  // pointer to the base object this render instance will be representing | Adds a new render instance to the Renderer that is attached to the passed in base object. Returns the index of that instance or -1 if failed. |
| Void | ChangeCamera | Bool bLock  // true for lock, false for unlock  RECT\* pLockRect  // rect to lock the camera to, if null will just lock to the screen area | Changes the camera mode to unlocked, locked to the screen size or locked to an area. |
| Void | Update | Float fElapsedTime  // time elapsed since last update | Updates all the render engines, uses time elapsed for things like Animations updates. |
| Void | Render |  | Calls render on the Renderer to render all the instances. |
| Void | ChangeState | Int nState  // id for new state that is now being used by the state system | The State System will call this function to tell the View that it’s now running a different game state. This will change how the View renders to the screen (if it should use the Renderer or tell the HudManager to render a menu). |
| Void | Shutdown |  | Cleans up everything. |

**Time to Complete Estimate**

* View Total 1 day
  + After all the other engines are done and working perfectly this will be just implementing a façade to the render engines and testing integration with the other systems like the Object Manager.

**Module Author(s)**

* Daniel Lima

Renderer

This module will contain most of the code specific the graphics API. It will have private members for the *D3D Object* and *Device* and take care of their initialization and clean up. All instances to be rendered will be within a vector inside this class and all of them will be rendered in the Render method, based on the camera information this class has access to. Since the other graphic engines won’t be able to directly access the *Device*, the Renderer will provide a public interface to allow for most of its functions to be used by other graphic modules like Render Mesh, Shader Manager and Effect Manager. Most of its methods will be called by the View class.

**Dependencies**

* Access to the following:
  + Direct3D9 libs
  + Camera
  + HUD Manager
  + Shader Manager
  + Mesh Manager
  + Texture Manager
  + Effect Manager
  + Render Instance
* Accessed by the following:
  + View (friend)
  + Shader Manager (through View)
  + Render Mesh (through View)
  + Effect Manager (through View)

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Bool | Initialize | HWND hWnd  // handle to the window we’re rendering to  Bool bWindowed  // true for windowed mode, false for full screen  Bool bVSync  // true for vsync on | Initializes Direct3D (object, device, effect pool) according to the passed in parameters. Returns true if succeeded, false otherwise. |
| Void | Render |  | Loops through all the instances, doing the render process for their shaders, textures and meshes. |
| Void | Update | Float fElapsedTime  // time elapsed since last update | Updates all the render instances, uses time elapsed for things like Animations updates. |
| Void | AddRenderInstance | RenderInstance\* pInst  // pointer to instance to be added to the renderer list | Pushes the passed in pointer to the vector of instances. |
| Camera\* | GetCamera |  | Returns a pointer to the camera so that it can be modified. |
| Void | CreateVertexIndexBuffer | IDirect3dVertexBuffer9\*\* pOutVertBuff  // vertex buffer this function will create  IDirect3dIndexBuffer9\*\* pOutIndexBuff  // index buffer this function will create  Void\* pVertSrc  // pointer to vert data to be copied  UINT uiVertLen  // number of bytes to copy from the vertSrc  Void\* pIndexSrc  // pointer to index data to be copied  UINT uiIndexLen  // number of bytes to copy from the indexSrc | Creates a vertex and an index buffer and copies the data over. |
| Void | Shutdown |  | Cleans up any allocated memory. |

**Associated Risks**

* Deferred rendering implementation will require some considered amount of research.

**Time to Complete Estimate**

* Renderer Total 7 days
  + Basic Forward Renderer 2 day
  + Research on deferred 1 day
  + Deferred Renderer 3 days
  + Testing 1 day

**Module Author(s)**

* Daniel Lima

Camera

This module will contain and manipulate the view and projection matrices that will be passed in to the shaders by the Renderer. It will normally follow a base object, but it may be locked to an area or just completely locked (no movement at all). It should only be directly accessed by the Renderer.

**Dependencies**

* Access to the following:
  + Player
* Accessed by the following:
  + Renderer

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Void | SetPlayer | Player\* pPlayer  // pointer to the object the camera will be following | Sets the player pointer for the camera and updates its position to be centered on the player. |
| Void | Update |  | Moves the camera based on the player position, if it’s not locked. |
| D3DXMATRIX | GetViewProjection |  | Returns the view matrix multiplied by the projection matrix. |
| Void | Lock |  | Makes it so the camera doesn’t move anymore. |
| Void | Unlock |  | Makes it so the camera can move again. |
| Void | SetArea | RECT\* pRect  // rect the camera will be able to move inside | Sets a limited area for the camera to be able to move. |
| Void | LookAt | D3DXVECTOR3 vPos  // position the camera will look at | Translates the camera so that the passed in point will be seen in the center of the screen. |

**Time to Complete Estimate**

* Camera Total 2 days
  + Basic camera behavior with camera locked – 1 day
  + Camera unlocked following the player – 1 day

**Module Author(s)**

* Daniel Lima

RenderInstance

Every object that will be rendered in game will be associated with a render instance. The render instance will basically contain an ID for the shader it’s using, which maps to the Shader Manager, an AnimationPlayer, which will contain all the information about the mesh and the animations being used, and a pointer to the game object associated to that instance. The Renderer will contain all the render instances.

**Dependencies**

* Access to the following:
  + Shader Manager
  + Animation Player
  + Base Object
* Accessed by the following:
  + Renderer
  + View

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Bool | Initialize | Int nMeshID  // id to the mesh this instance will be using  BaseObject\* pObj  // pointer to the base object this instance will be associated with | Creates an instance based on the passed in information. Returns true if succeeded, false otherwise. |
| Void | Update | Float fElapsedTime  // time elapsed since last update | Updates the instance’s animation player based on the elapsed time. |
| Void | PlayAnimation | Int nAnimID  // id to the animation to play  Bool bLooping  // true if the animation should loop | Tells the animation player contained in this class to play an animation. |
| Void | Stop |  | Tells the animation player contained in this class to stop the current animation. |

**Time to Complete Estimate**

* Render Instance Total 2 days
  + Basic functionality – 1 day
  + Integration and Testing – 1 day

**Module Author(s)**

* Daniel Lima

MeshManager

The Mesh Manager will be the container for all the meshes. By calling the LoadMesh function, the View will be able to load a mesh into this container and receive an ID that maps to its index on the array. This ID will later be passed in by the Renderer to any function that has to deal directly with the meshes. It will be impossible to actually access the RenderMesh objects outside of this class.

**Dependencies**

* Access to the following:
  + Render Mesh
* Accessed by the following:
  + Animation Player
  + Renderer

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Int | LoadMesh | Char\* szFilename  // name of the mesh file to load | Creates a new mesh, calls Load on it with the passed in file name, pushes it to the vector of meshes and returns the index. Returns -1 if failed. |
| Void | DrawMesh | Int nMeshID  // id of the mesh to draw | Calls Draw on the mesh of the passed in ID. |

**Time to Complete Estimate**

* MeshManager Total 1 day
  + The mesh specific code to load and draw will actually be on the Render Mesh class.

**Module Author(s)**

* Daniel Lima

RenderMesh

This module will contain all the geometry-related information for a given mesh, such as the vertex and index buffers, the vertex declaration, the number of vertices and primitives, etc. It will also contain IDs for textures in the TextureManager that this mesh is using. In the Load function, all of the information a mesh needs will be loaded from a file exported by Maya. This will include texture information so the RenderMesh will call the TextureManager LoadTexture function and store the returned IDs. The MeshManager will contain a vector of RenderMesh objects.

**Dependencies**

* Access to the following:
  + View
  + Texture Manager
* Accessed by the following:
  + Mesh Manager

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Bool | Load | Char\* szFilename  // name of the mesh file to load | Loads static mesh data like vertices (positions, normals and uvs), indices and textures from the passed in file. Returns true if succeeded. |
| Void | Draw |  | Sets the mesh data (vertex declaration, stream source, index source) to be rendered. |

**Time to Complete Estimate**

* RenderMesh Total 3 days
  + Load 1 ½ day
  + Draw ½ day
  + Testing and integration 1 day

**Module Author(s)**

* Daniel Lima

TextureManager

The Texture Manager will be the container for all the textures. By calling the LoadTexture function, other engines will be able to load a texture into this container and receive an ID that maps to its index on the array. This ID will later be passed in to any function that has to deal directly with the textures. It will be impossible to actually access the textures outside of this class.

**Dependencies**

* Access to the following:
  + View
  + Shader Manager
* Accessed by the following:
  + Render Mesh
  + Renderer

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Int | LoadTexture | Char\* szFilename  // name of the texture file to load | Creates a texture, loads it from the file name, pushes it to the vector and returns the index. Returns -1 if failed. |
| Void | ApplyTexture | Int nShaderID  // id for the shader to be used  Int nTextureID  // id for the texture to be used | Sets the texture of the passed in ID to the passed in shader. |

**Time to Complete Estimate**

* Texture Manager Total 1 day

**Module Author(s)**

* Daniel Lima

ShaderManager

The Shader Manager will be the container for all the shaders. By calling the LoadShader function, other engines will be able to load a shader into this container and receive an ID that maps to its index on the array. This ID will later be passed in to any function that has to deal directly with the shaders. It will be impossible to actually access the shaders outside of this class.

**Dependencies**

* Access to the following:
  + View
* Accessed by the following:
  + Render Instance
  + Renderer
  + Texture Manager
  + View

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Int | LoadShader | Char\* szFilename  // name of the shader to load | Creates a shader, loads it from the file name, pushes it to the vector and returns the index. Returns -1 if failed. |
| UINT | BeginShader | Int nShaderID  // id for the shader to be used | Calls Begin on the shader of the passed in ID and returns the number of passes or -1 if failed. |
| Void | EndShader | Int nShaderID  // id for the shader to be used | Calls End on the shader of the passed in ID. |
| Void | BeginPass | Int nShaderID  // id for the shader to be used  UINT uiPass  // number of pass to start | Calls BeginPass on the shader of the passed in ID passing uiPass. |
| Void | EndPass | Int nShaderID  // id for the shader to be used | Calls EndPass on the shader of the passed in ID. |
| Void | SetTechnique | Int nShaderID  // id for the shader to be used  Char\* szTechnique  // technique name to set | Accesses the shader of the passed in ID and sets its technique to the passed in szTechnique. |
| Void | SetMatrix | Int nShaderID  // id for the shader to be used  D3DXHANDLE hParam  // handle to the uniform to set  D3DXMATRIX\* pMatrix  // matrix to set the uniform to | Accesses the shader of the passed in ID and sets the matrix hParam to the passed in pMatrix. |
| Void | SetTexture | Int nShaderID  // id for the shader to be used  D3DXHANDLE hParam  // handle to the uniform to set  IDirect3DTexture9\* pTexture  // texture to set the uniform to | Accesses the shader of the passed in ID and sets the texture hParam to the passed in pTexture. |
| Void | CommitChanges | Int nShaderID  // id for the shader to be used | Calls CommitChanges on the shader of the passed in ID. |

**Time to Complete Estimate**

* ShaderManager Total 1 day

**Module Author(s)**

* Daniel Lima

EffectManager

The Effect Manager will be the container for all the effect types and the active emitters in the game. By calling the LoadEffect function, other engines will be able to load an effect into this container and receive an ID that maps to its index on the array. This ID will later be passed in to the Create function to make emitters based on that effect type. It will be impossible to actually access the emitters or effects outside of this class.

**Dependencies**

* Access to the following:
  + Effect
  + Emitter
* Accessed by the following:
  + View
  + Renderer

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Int | LoadEffect | Char\* szFilename  // name of the effect file to load | Creates an effect, loads it from the file name, pushes it to the vector and returns the index. Returns -1 if failed. |
| Void | CreateEmitter | Int nEffectID  // id for the effect to be used  BaseObject\* pObj  // pointer to the object this will be emitting from  Float fLife  // lifetime of the emitter  // 0 = one time  // -1 = infinite | Creates a new emitter associated to the passed in Base Object that will spawn an instance of the effect of the passed in ID. |
| Void | Update | Float fElapsedTime  // time elapsed since last frame | Updates all the emitters based on the elapsed time. |
| Void | Render |  | Loops through all the emitters and their active particles and renders them. |

**Time to Complete Estimate**

* EffectManager Total 4 days
  + Effect class 1 day
  + Emitter class 1 day
  + EffectManager class 1 day
  + Testing and integration 1 day

**Module Author(s)**

* Daniel Lima

HudManager

While the Renderer will be the class responsible to render all game objects and everything 3D, the HUD Manager will be the one to trust for everything that should be rendered on top of the regular game graphics or any 2D menu in general. For that it will make use of D3D sprites and the Font Manager. The View module will handle which one of them should be used based on the game state.

**Dependencies**

* Access to the following:
  + Renderer
  + Font Manager
* Accessed by the following:
  + View

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Void | Initialize |  | Loads all the fonts and sprites that the module will ever need to render. |
| Void | Render | Int nState  // id for the state that should be rendered | Renders the HUD/menu based on the game state. |

**Time to Complete Estimate**

* HUD Manager Total 4 days
  + Research on 2d/3d graphics integration 1 day
  + Implementation 2 days
  + Testing and integration 1 day

**Module Author(s)**

* Daniel Lima

FontManager

The Font Manager will be the container for all the bitmap fonts used in menus and the in-game head up display. It will be part of the HudManager and provide an interface for it to be able to load new fonts and write text to the screen with them.

**Dependencies**

* Access to the following:
  + Renderer
  + Bitmap Font (struct)
* Accessed by the following:
  + Hud Manager

**Public Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Return** | **Name** | **Parameters** | **Description** |
| Int | LoadFont | Char\* szFilename  // name of the effect file to load | Creates a bitmap font, loads it from the file name, pushes it to the vector and returns the index. Returns -1 if failed. |
| Void | Write | Int nFontID  // id of the font to use  String szText  // text to print to the screen  RECT\* pArea  // area to print text  Float fScale  // scale of the text | Uses the Renderer to write text to the screen bases on the passed in parameters. |

**Time to Complete Estimate**

* FontManager Total 2 days
  + Implementation 1 day
  + Testing and integration 1 day

**Module Author(s)**

* Daniel Lima