HONG KONG INSTITUTE OF VOCATIONAL EDUCATION

Laboratory 8: 3D Models

Module Intended Learning Outcome (#2):
On completion of the module, students are expected to be able to:

- develop 2D and 3D graphics programs for general gaming purposes;

Lesson Intended Learning Outcome:
On completion of this lab, students are expected to be able to:

- Use 3D models as game objects in a 3D gaming environment.

<u>TASK 1 – Start With the Ground</u>

Let's start with a new Monogame project.

1. Copy all models and the related texture files and folders into your project's **Monogame Content Pipeline Tool.** You may want to store the files into a subfolder "**Models**" under the **Content** folder.
2. For those files which are not models (.x and .fbx files), check their properties and change the Build Actions to **Copy**.
3. Add a class-level **Model** variable to store the ground model. Load the ground model to its variable in ***LoadContent()*** function.

4. Add two class-level **Matrix** variables to store the view transformation and projection matrices. Inside your ***Initialize()*** function, initialize the two matrices. Set the camera at (0, 0.9, 0), looking at (0, 0.9, -0.5) with the up direction as Vector3.Up. Set the field of view angle as 45°, frustum's near and far clip plane at z = 0.1f and 100f respectively. The aspect ratio should be obtained through the viewport.

5. Also inside your ***Initialize()*** function, set the **GraphicsDevice** status for drawing the model:

```
GraphicsDevice.BlendState = BlendState.Opaque;
GraphicsDevice.DepthStencilState = DepthStencilState.Default;
GraphicsDevice.SamplerStates[0] = SamplerState.LinearWrap;
```
6. Call model's ***Draw()*** function inside the ***Draw()*** function of *Game1.cs*. The ground model is scaled down to show a better visual result.
```
groundModel.Draw(Matrix.CreateScale(0.001f), view, project);
```

<u>TASK 2 – Add the Windmill</u>

1. First, we will build a **DrawableGameComponent** for the **Windmill** object. Choose **Class** from **Visual C#** to create a class. Add appropriate using statements and modify the class header to inherit **DrawableGameComponent**.
2. The windmill consists of two individual models: base and fan. Declare two class-level **Model** variables for them.
3. Also, add a class-level **Vector3** variable to hold the position, two **float** variable to store the fan rotation angle and speed. The default speed of the fan rotation is 0.02f per update. Finally add two **Matrix** variables to store the view and projection matrices. As the two matrices are created and set in *Game1.cs*, we need to give the **Game1** class access to these two variables. Use the access modifier **internal** instead of **public**, which means the variables are accessible by the classes inside the project only.
```
  internal Matrix view, project;
```

4. Modify the constructor of **Windmill** so that it also reads in Vector3 position which will store in the variable declared above.
   ```
   public Windmill(Game game, Vector3 pos) : base(game) { position = pos; }
   ```
5. Inside the ***LoadContent()*** function, load the two windmill models.
6. Inside the ***Update()*** function, update the rotate angle by adding the speed to it. Make the angle always between 0 and 2π.
7. Inside the ***Draw()*** function, declare a local **Matrix** variable for world transformation.
8. We will draw the base model first and then the fan. Before drawing each model, we need to calculate for the world transformation matrix.

   The world transformation matrix for the base is simple: scale up the model to 10f and translate it to its position.

   The world transformation for the fan needs one more component: rotation of the fan leaves. Rotate the model along the Z-axis according to the stored angle variable.

   Use model's ***Draw()*** function, with the calculated world transformation matrix and stored view and projection matrices will get the work done.
   ```
     xxxModel.Draw(world, view, project);
   ```
9. The final steps will be the modification to be made on *Game1.cs*. You need to add a class-level variable for the **Windmill** and initialize it in the ***Initialize()*** function. Place the windmill at the position (0, 0.9f, -4.0f). Remember to set the view and projection matrices to the windmill variable. Also, you need to add the windmill to the **Components** list of the game.
   ```
     windmill = new Windmill(...);
     windmill.view = view;
     windmill.project = project;
     Components.Add(windmill);
   ```

TASK 3 – Add an Orbiting Helicopter

In the third part of this lab, we will build a **DrawableGameComponent** class **Helicopter** and instantiate a **Helicopter** object that flies around the windmill in a "tilted" mode.

1. Add a new **Class Helicopter** to the MonoGame project. Make it inherit the **DrawableGameComponent** class and add appropriate using statements so that it is capable to use XNA framework.
2. We have only one model in the **Helicopter** class. Create a **Model** variable for the model. Other attributes for the **Helicopter** class includes: rotate center (the **Vector3** position of the windmill), rotate angle along Y and Z axis (**float**), rotate speed (**float,** initial value = 0.05f), and the initial offset from the center (**Vector3,** initial value = (0, 0, -1), i.e. 1 unit in front of the windmill). Remember to add **internal Matrix** variables for the view and projection matrices also.
3. As in Task2, modify the constructor so that it accepts a Vector3 parameter as the center of rotation.
   ```
   public Helicopter(Game game, Vector3 pivot) : base(game) { rotateCenter =
   pivot; }
   ```
4. Inside the ***LoadContent()*** function, load the corresponding model for the helicopter.
5. Inside the ***Update()*** function, update the rotate angle by adding the speed to it. Make the angle always between 0 and 2π.
6. Inside the ***Draw()*** function, calculate the world transformation matrix and call the model's ***Draw()*** function to render the helicopter.

The world transformation matrix consists of scaling (0.05f), rotation along Y-axis to fix the model direction (90°), orbit translation (according to the offset), orbit rotation along Y-axis (rotate angle) and Z-axis (constant tilt angle of 30°) and the final translation (to the rotate center, i.e. the windmill position).

7. To complete the task, add a **Helicopter** object in your *Game1.cs*. Initialize it with the windmill position. Pass the view and projection matrix to corresponding variables in the **Helicopter** object and add it to the Game's **Components** list.