HONG KONG INSTITUTE OF VOCATIONAL EDUCATION
Laboratory 9: 3D Chase Camera

Module Intended Learning Outcome (#2 and #5):
On completion of the module, students are expected to be able to:
- develop 2D and 3D graphics programs for general gaming purposes;
- identify and apply appropriate user input methods for graphics and/or game programs.

Lesson Intended Learning Outcome:
On completion of this lab, students are expected to be able to:
- Interact with 3D game objects using user input;
- Apply 3D chase camera in a 3D gaming environment.

TASK 1 – Adding a Camera

Let's add a camera object into your project (Lab 8). We will use the **ChaseCamera** class given in the lecture example.

1. Remove the **Matrix** variables ***view*** and ***project***. We will use the view transformation and projection matrices inside the camera instead.
2. Download the sample code from moodle if you haven't done so. Copy the *ChaseCamera.cs* into your project. You will need to modify the namespace tag of the **ChaseCamera** class to the namespace of your current project.

3. Add a class-level **ChaseCamera** variable to your project. Initialize your variable by these settings:

```
cam.up = Vector3.Up;
cam.fieldOfView = MathHelper.PiOver4;
cam.aspectRatio = GraphicsDevice.Viewport.AspectRatio;
cam.nearClip = 0.1f;
cam.farClip = 1000;
cam.positionOffset = new Vector3(0, 0, -1); // cam position relative to
windmill
cam.lookAtOffset = Vector3.Zero; // cam look at position relative to windmill
cam.targetUp = Vector3.Up;
cam.targetPosition = new Vector3(0.0f, 0.9f, -4.0f); // windmill position
cam.targetDirection = -Vector3.Forward;
```

Remember to add your camera variable to the Components list of your game.
4. Inside the ***Update()*** function, pass your camera's view and projection matrix to the windmill and the helicopter.
5. Inside the ***Draw()*** function, use the camera's view and projection matrix for drawing the ground model.
6. Compile and run the project. The result will look the same as the one before.

TASK 2 – Adding an Airship that Accepts User Input

1. Add a new class **HelicopterControllable** that inherits **DrawableGameComponent**. This helicopter is similar to the one we have done in Lab 8, but with the following attributes instead:
```
internal Vector3 direction, position, right, up; // additional directions is
added
Vector3 initPosition; // stores the initial position, for resetting the
position
Model model;
internal ChaseCamera cam; // variable to store the chase camera
```

```
    float yaw, pitch, speed; // rotate angle along y and x axes, and the speed of
    airship
    const float pitchRate = 0.01f;
    const float yawRate = 0.005f;
    const float forwardSpeed = 0.025f;
```

2. In the class constructor, change the function header so that it accepts an additional **Vector3** parameter ***pos***, which will be stored in both the variables ***position*** and ***initPosition***. Also, initialize the ***up*** and ***direction*** variables to **Vector3.Up** and **-Vector3.Forward** respectively. Calculation your ***right*** Vector using **Vector3.Cross(*direction, up*)**.
3. In the ***LoadContent()*** function, load and store the model for the class.
4. In the ***Update()*** function, you will need to get the keyboard state and response to certain input accordingly.

	Key Pressed	**Action**
Setting the yaw angle	Left	Set yaw angle to the constant yaw rate.
	Right	Set yaw angle to ***negative*** constant yaw rate.
	Otherwise	Set yaw angle to 0.0f
Setting the pitch angle	Up	Set pitch angle to the constant pitch rate.
	Down	Set pitch angle to ***negative*** constant pitch rate.
	Otherwise	Set pitch angle to 0.0f
Setting the airship speed	Space	"Turbo" activated – set speed to 4x constant speed.
	Otherwise	Normal speed – set speed to constant forward speed.

5. After handling the inputs, recalculate the three Vectors: ***direction***, ***up***, ***right*** and ***position***.
   - The transformation matrix is calculated by the following formula:
     ```
     Matrix transform = Matrix.CreateFromAxisAngle(right, pitch) *
         Matrix.CreateRotationY(yaw);
     ```

     The rotations yaw and pitch are along the world's up and helicopter's "right" axes.
   - ***Up*** and ***direction*** is done by transforming the two vectors according to the above yaw and pitch angle.

     ```
     up = Vector3.TransformNormal(up, transform);
     direction = Vector3.TransformNormal(direction, transform);
     ```
   - The two vectors should be normalized (reduced to unit vectors) before the ***right*** vector is recalculated again. The ***up*** vector is then recalculated again using Cross Product to ensure the orthogonality is kept.
   - ***Position*** is then calculated using the ***direction*** and ***speed***.
     ```
     position += direction * speed;
     ```
   - Keep the helicopter above the ground by setting back the Y-coordinate of its position to 0 once it drops below the ground level.
6. Add one more statement in ***Update()*** so that when "R" is pressed, the ***position***, ***up*** and ***direction*** of the airship will be reset to its initial values.
7. Add the following private function that calculates the rotation matrix for the helicopter according to its facing direction:

```
private Matrix YDirection() {
  Matrix rotationY = Matrix.Identity;
  rotationY.Forward = direction;
  rotationY.Up = up;
  rotationY.Right = right;
  return rotationY;
}
```

8. In the ***Draw()*** function, construct your world transformation function accordingly (***scaling***, ***rotation*** and ***translation***) and call model's ***Draw()*** function to render your helicopter onto the screen.
9. As the final step, add the new **HelicopterControllable** as a variable in class-level and initialize it to the position (0,1,-6). Remember to store the camera variable into your **HelicopterControllable** object and add the helicopter into your game's **Components** list.

TASK 3 – Setting the Chase Camera

The current "chase camera" does not chase anything. It just keeps staring at the windmill. Let's modify it so that it flies behind your new helicopter.

1. As your helicopter now has a "pitch" element, which does not exist in the lecture example, you will need to add one more attribute to the **camera** to keep track on the helicopter's up direction:
   ```
   internal Vector3 targetUp;
   ```
2. Inside your ***UpdatePosition()*** function, you should update your camera ***direction*** and ***up*** direction before any other update is done.
   ```
   direction = targetDirection;
   up = targetUp;
   ```
3. In your *Game1.cs*, modify the initial setting of the camera accordingly.
   ```
   cam.positionOffset = new Vector3(0, 0.25f, 1);
   cam.lookAtOffset = new Vector3(0, 0.1f, -1);
   ```
4. In the ***Update()*** function in *Game1.cs*, you need to update the three "target" related vectors of the camera.
   ```
   cam.targetDirection = player.direction;
   cam.targetPosition = player.position;
   cam.targetUp = player.up;
   ```
5. Compile and run the program. Now, your camera should go behind your helicopter and fly following it.