HONG KONG INSTITUTE OF VOCATIONAL EDUCATION
Laboratory 10: Shooting 3D Rockets

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 Ray Tracing Technique to detect 3D collisions.

TASK 1 – Shooting Rockets

While you are enjoying flying your helicopter in Lab 9, let's add some rockets and shoot them out.

1. Copy the **Rocket** class (*Rocket*.cs) from the sample code from the lecture notes to your project. Remember to modify the namespace tag of the **Rocket** class to the namespace of your current project.
2. Modify the ***Update()*** function, so that the rocket will be affected by gravity immediately after it is shot.
3. As in the sample code, declare a class-level **Rocket** array. Initialize your array with 10 rockets. Create each rocket, add each of them to the **Components** list and set the camera object to the rockets.
4. In *Game1.cs*, add a **KeyboardState** variable ***previousKb*** to store the keyboard state after each update. Initialize it to the current **KeyboardState**.
   ```
   KeyboardState previousKb = Keyboard.GetState();
   ```
5. Inside the ***Update()*** function, you should check if the user pressed the "S" key. It is counted as a valid key only if the previous keyboard state does not have "S" key pressed.
   ```
   KeyboardState kb = Keyboard.GetState();
   if (kb.IsKeyDown(Keys.S) && previousKb.IsKeyUp(Keys.S)) { ... }
   ```
6. Launch a rocket if "S" key is pressed. Loop through the array to find the first inactive rocket. The initial direction and position of your rocket will be the direction and position of your **HelicopterControllable** object.
7. Compile and run the project to see if the rocket comes out.

TASK 2 – Check for Collision

1. Add a sprite font to your project. Add the sprite font to your **Content** folder and remember to add a **SpriteFont** object in *Game1.cs*.
2. Add a class-level **string** variable ***message*** to store the message to be displayed in the ***Draw()*** function of *Game1.cs*.
3. In your ***Update()*** function, cast a ray for each active rocket inside a ***for*** loop, using rocket's previous position and current position:
   ```
   for (int i = 0; i < 10; i++) {
     if (rockets[i].active) {
           Ray ray = new Ray(rockets[i].previousPosition,
                 Vector3.Normalize(rockets[i].position -
   rockets[i].previousPosition));
           // Other code ...
   ```

4. Then, calculate the distance travelled by the rocket in that frame. The distance can be found by multiplying the length of the speed vector by the elapsed time of the frame.

```
float distance = rockets[i].speed.Length() *
(float)gameTime.ElapsedGameTime.TotalSeconds;
```

5. The ***Update()*** function will then check if the ray intersects the bounding sphere of any 3D objects in the game. In this exercise, we will check against the windmill base, the windmill fan and the orbiting helicopter.

   To check against the **BoundingSphere** of each 3D object, you will need to loop through the meshes of the model of that 3D object. You may need to add **internal** access modifier to the **Model** attribute of your 3D game components class so that we can access them in *Game1.cs*.

   Move the **BoundingSpheres** of your model meshes to the model's position and scale them accordingly (See below). Use the ray created in Step 3 to see if the ray intersects the sphere and if the hit distance is shorter than the travel distance. Is so, store the result in the ***message*** string.

   The following shows the checking for the windmill fan. You will need to work out the other parts of the game.

```
foreach(ModelMesh mm in windmill.fanModel.Meshes) {
  BoundingSphere sphere = mm.BoundingSphere;
  sphere.Center += windmill.position; // move to position of your fan model
  sphere.Radius *= 10f; // model scale
  if (ray.Intersects(sphere) != null && ray.Intersects(sphere) < distance) {
      message = "Bullet " + (i + 1) + " will hit the fan.";
      rockets[i].active = false;
  }
}
```


   The above example moves the center and scales to the radius directly. Alternatively, you may transform your **BoundingShpere** to the appropriate place using the world transformation matrix of that model (e.g. the orbiting helicopter). You may add/change the variable declaration to facilitate your work.

6. Add lines of statements to print out the message using the sprite batch inside the ***Draw()*** function. To draw strings (or other 2D objects) in a 3D game, we need to setup different options in the ***spriteBatch.Begin()*** function call.

```
spriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend,
  SamplerState.LinearWrap, DepthStencilState.None, RasterizerState.CullNone);
```

   Print out message at the top-left corner of the screen and ends the ***spriteBatch*** session. You need to restore the **BlendState** and the **DepthStencilState** setting after drawing the 2D sprites.

```
GraphicsDevice.BlendState = BlendState.Opaque;
GraphicsDevice.DepthStencilState = DepthStencilState.Default;
```

7. Compile and run the game. Discuss what the problem of using bound spheres inside the model is. Give some suggestions to improve the result.