CS 3920/5920 - Computer Graphics Fall 2017

PROGRAM: 5: 3D Space Game

POINTS: 25

DUE DATE: December 11

GRACE DATE: December 14 (demonstrated no later than 5:00 p.m.)

-----------------------------------------------------------------------------------------------------------------

The default for this program is a “first-person” spaceship roaming around shooting at space garbage.

You must put the project files in a Prog5 folder in your group share folder. Sign up for a 15 minutes demo time.

I assume both individuals will contribute approximately equal to this project and both will understand the pieces. I can ask questions during the demo and if it is clear that an individual knows "less than they should" about the project, then they can expect point loss.

------------------------------------------------------------------------------------------------------------------

You must use the classes from program 4 as a framework to make a 3D space-type game. The minimum requirements are:

1. You must have several “cleverly-moving” figures. For the default, this would mean loading a bunch of figures from a folder similar to program 3 and having them move as in Program 3.
2. You must use a **Spotlight**. You must allow different colors for the spotlight like you did for the light in Program 4.
3. The game must have an objective and some kind of scoring. Statistics, positions, relevant game information, etc. must be displayed. It must have an "easy" mode (level) where things are easy to see and do. In the easy mode, I don't want tiny things far away that are very hard to navigate to and shoot. I am not implying you need different levels, but there must be at least one easy level that I can play and understand.
4. You must display the position and direction of the spaceship.
5. You must have some kind of "Help" available that explains the game and how to play it. It can be as simple as a dialog that pops up when Help is selected from a menu.
6. The player can control the spaceship using the keyboard and/or mouse.
7. You must encapsulate when it makes sense to do so. For example, if your game has a spaceship manipulated by the player, you must have a spaceship class.
8. Don't make the user open a folder to load figures. Have the program open them itself. Put them in a subfolder under the executable. You must use a relative path name!
9. As always, I expect good, clean OOP. Use polymorphism and inheritance where appropriate.

I am not requiring you use any other light than a spotlight and ambient, although you can add others if you want. The Spot Light can replace the current point light from Program 4. The only new uniform variable needed is a SpotDirection (vec3) assuming you use fixed values in the vertex shader program for the Spot Angle and Spot Cutoff. Having them in the Vertex Shader program is okay, since if you want to change them while testing and experimenting, you can edit the shader program file and don’t need to recompile your C# program.

The required shader code equations can be derived from the LighingOpenGL\_Redbook.doc I put in the 1Materials folder. Assuming an added uniform variable SpotDirection and local shader program variables for spotCutOffAngle (I suggest experimenting with values from 5 to 60) and spotExponent, the shader code to calculate the SpotEffect is below. You **MUST** comment each line in the shader program to show you understand. Use LighingOpenGL\_Redbook.doc to help you do this.

vec3 spotLightDir = normalize(vec3(ViewMatrix \* vec4(SpotDirection,0.0)));

float sDotSpotDir = max( dot(-s, spotLightDir), 0 );

float cosSpotAngle = cos(radians(spotCutOffAngle));

float spotEffect = 0;

if (sDotSpotDir >= cosSpotAngle)

spotEffect = pow( sDotSpotDir, spotExponent );

Then for diffuse and specular, you multiply this into what you had for Program 4 for those (again, see LighingOpenGL\_Redbook.doc). This must **NOT** be part of the ambient. The ambient stays as:

vec3 ambient = vec3(GlobalAmbient) \* VertexColor;

Your program must still allow the Ambient to be set in the C# program. My thought is that we would set Ambient fairy low, so you could see things, but not very clearly. As you hunt a piece of space garbage, you will turn the spaceship to face it, the spot light will light it up, then you shoot at it.

------------------------------------------------------------------------------------------------------------------

Some hints based on what I did. These may or may not be applicable to your program.

1. I added a "copy constructor" to Figure: public Figure(Figure f) that initialized all the data members using those from f. I used this so that I could add projectiles to the Projectile list when the Fire button is pressed without re-reading from a file.
2. I added two CollidesWith methods to Figure. One is for collisions between figures, the other for collisions between a figure and the Ship (or any "box" at a given position). Make sure you understand them - draw simple pictures with two cubes. It does not account for rotations, but it is close enough for this program.

public bool CollidesWith (Vector3 OtherObjectMax, Vector3 OtherObjectMin,

Vector3 OtherObjectPosition)

{

for (int i = 0; i < 3; i++)

{

if ((max[i] + translateAmount[i]) <

(OtherObjectMin[i] + OtherObjectPosition[i]) ||

(OtherObjectMax[i] + OtherObjectPosition[i]) <

(min[i] + translateAmount[i]))

return false;

}

return true;

}

public bool CollidesWith(Figure f)

{

return CollidesWith(f.max, f.min, f.translateAmount);

}

1. I added Collision checkers to the FigureList. For example, one was for checking projectiles against garbage: public void CheckCollisionsKillIfDetected (FigureList list).

It was a double-for loop that removed colliding objects if an object from one list collided with an object from the other list (e.g, a projectile colliding with a piece of space debris). I suggest that the outer loop goes backwards (know why!). Also, you can use "break" to break out of the inner loop on a collision, since you may remove both the garbage and projectile from their lists. Or maybe you have to hit it X times? But sooner or later, you remove it. You could also add one for collisions between the Ship & garbage, one for garbage against garbage, and anything else appropriate.

1. I added an Add method to FigureList:

public void Add(Figure fig, MovePattern movement)

1. You can experiment with having the keyboard handler on the Form or on the GL control. If you put it on the Form, handle the KeyPress or KeyDown event of the Form, and set the KeyPreview property of the Form to true so that keyboard messages are received by the Form before they reach any controls on the Form.

------------------------------------------------------------------------------------------------------------------

Extra Credit: I will give up to 5 points extra credit for "extras" beyond what I would consider a good job on this program, on an "as it strikes me" basis. To be eligible for the extra credit, you must **complete** your demo before 4:00 p.m. on the DUE DATE!

------------------------------------------------------------------------------------------------------------------

**Default Project: Description and Hints**

You will add a “first-person” (no figure image for it) space ship that moves around and hunts space garbage. The space garbage is stored in a FigureList object (as in Program 3). The ship is primarily manipulated with the keyboard but you can also add a mouse if you want. The spaceship fires projectiles at the space garbage. The projectiles will be stored in a separate FigureList object. Note: you will have only one FigureList class but at least two lists of figures (two instances of the FigureList class), one for the moving garbage and one for the moving projectiles. The garbage must move around cleverly, like in program 3. The projectiles will most likely move in a straight line from the space ship in the direction the ship is pointing. I made a MoveStraight MovePattern with constructor:

public MoveStraight(Vector3 direction, double moveAmount)

and translated the projectile figure to the ship's position before adding it to the projectile's FigureList. The Move method would just translate by the moveAmount\*moveDirection for each X, Y, Z component.

------------------------------------------------------------------------------------------------------------------

You must make a singleton for the Ship class. I have a sample below you could finish. However, I assumed the spherical coordinates phi and theta for movement. You are not required to use those and can change those parts of the class. The Theta corresponds to turning left or right (yaw) and the Phi corresponds pointing up or down (pitch). Note that I don’t have a “roll”; however, you can add that if you want. You can modify the Ship class, but in any case, add more comments to show you understand what is being done. Recall how singleton works, e.g.: Ship.Instance.Move(0.5);

// Uses spherical coordinates with a unit sphere as follows:

// y = cos(phi); x = sin(phi) \* sin(theta); z = sin(phi) \* cos(theta);

public class Ship

{

private float dirTheta; // Theta for current direction

private float dirPhi; // Phi for current direction

private Vector3 direction; // Unit vector for direction ship is pointing

private Vector3 position; // Current position of ship

private Vector3 max, min; // Bounding Box of ship

private static Ship \_instance = null; // Singleton instance – must be private!

public static Ship Instance

{

get

{

if (\_instance == null)

\_instance = new Ship();

return \_instance;

}

}

public Vector3 BoundingBoxMax { get { return max; } }

public Vector3 BoundingBoxMin { get { return min; } }

public Vector3 Position { get { return position; } }

public Vector3 Direction { get { return direction; } }

private Ship()

{

Reset();

min = new Vector3(0, 0, 0);

max = new Vector3(1, 1, 1); // Make it whatever size you want

// Could have other things

}

public void Reset()

{

position = new Vector3(25, 25, 25); // Start it wherever you want.

// Looks back at origin. You can have it point wherever you want.

dirPhi = (float)Math.Acos(-position.Y / position.Length);

dirTheta = (float)Math.Atan2(-position.X, -position.Z);

ChangeDirection(0, 0);

}

public void ChangeDirection(float deltaTheta, float deltaPhi)

{

dirTheta += deltaTheta;

dirPhi += deltaPhi;

direction[1] = (float)Math.Cos(dirPhi);

direction[0] = (float)(Math.Sin(dirPhi) \* Math.Sin(dirTheta));

direction[2] = (float)(Math.Sin(dirPhi) \* Math.Cos(dirTheta));

}

public void Move(double amount)

{

// Change "position" by moving "amount" in "direction"

for (int i = 0; i < 3; i++)

position[i] += (float)(amount \* direction[i]);

}

// Main form can call this to get the View Matrix

public Matrix4 LookAt()

{

return Matrix4.LookAt(position[0], position[1], position[2],

(float)(position[0] + Math.Sin(dirPhi) \* Math.Sin(dirTheta)),

(float)(position[1] + Math.Cos(dirPhi)),

(float)(position[2] + Math.Sin(dirPhi) \* Math.Cos(dirTheta)),

0, 1, 0);

}

// etc. Add whatever you need

}