Computer Graphics

Project Submission

Target Practice

Submitted By:

Parth Kapoor (102203739)

Giriraj Dhyani (102203875)

Yash Kukshal (102203543)

Submitted To:

Dr. Nisha Thakur

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

Introduction

This project, titled Aim and Shoot, is an interactive 3D shooting simulation implemented in OpenGL 3+. Users can aim and fire three types of projectiles (bullet, grenade, and shuriken) at a target inside a 3D room. The application demonstrates core computer graphics principles, including scene modeling, camera control, geometric transformations, curve-based trajectories, collision detection, and basic lighting. Through hands-on implementation, the project meets the course objectives of understanding graphics primitives, 2D/3D transformations, visible surface detection, illumination models, and curve design.

Some Computer Graphics Concepts Used

1. **Graphics Primitives and Modeling**

   Use of OpenGL quadric objects and GLUT primitives (cylinder, disk, torus, sphere, cube) to construct bullets, grenades, shuriken, walls, and room surfaces.

   Polygonal modeling with glBegin(GL_POLYGON) for walls and shuriken parts.

2. **2D & 3D Geometric Transformations**

   Translation, Rotation, Scaling: Applied via glTranslated, glRotated, and glScaled to position and orient objects.

   Composite Transformations: Combined rotations and translations to achieve firing trajectories and object hierarchies (push/pop matrix stack).

3. **Curve Design (Bezier Curves)**

   Parametric cubic Bezier curve for grenade and shuriken trajectories.
   Control points define height arc and landing position, calculated in bezier()

4. **3D Viewing and Projection**

   Perspective projection with gluPerspective(80, aspect, 0.001, 1000).

   Camera positioning using gluLookAt for both player view and replay camera.

5. **Visible Surface Detection & Collision**

Simple depth-based collision: hitWall and hitTarget functions test object coordinates against world bounds and target radius.

6. **Illumination and Surface Rendering**

   Two light sources: directional and spotlight, configured in setupLights().

   Material properties (ambient, diffuse, specular) for basic Phong illumination.

User Defined Functions

Modeling

void createBullet(character* thisCharacter);

void createGrenade(character* thisCharacter);

void createShurikenPart();

void createShuriken(character* thisCharacter);

void createWall();

void createRoom();

void createTarget(scoreBoardTarget* target);

Gameplay and Game Loop

void initGame();

void endGame();

void drawGame(scoreBoardTarget* target, character* character);

void calculateScore(character* character);

void replay();

void replayFiringCamLogic();

Motion and Firing Logic

void changeCharacterTrajectoryAimLogic(int direction);

void fireCharacter();

void fireBullet(character* bulletCharacter);

void fireGrenade(character* grenadeCharacter);

void fireGrenadeStart(character* grenadeCharacter);

void fireGrenadeLogic(character* grenadeCharacter);

void fireShuriken(character* shurikenCharacter);

void fireShurikenStart(character* grenadeCharacter);

void fireShurikenLogic(character* grenadeCharacter);

int* bezier(float t, int* p0, int* p1, int* p2, int* p3);

Replay Handling

void characterHit();

Input and I/O

void passM(int x, int y);

void keyUp(unsigned char k, int x, int y);

Environment Configuration

void setupCamera();

void setupLights();

OpenGL Callback Functions

void Display();
void anim();

SCREENSHOTS

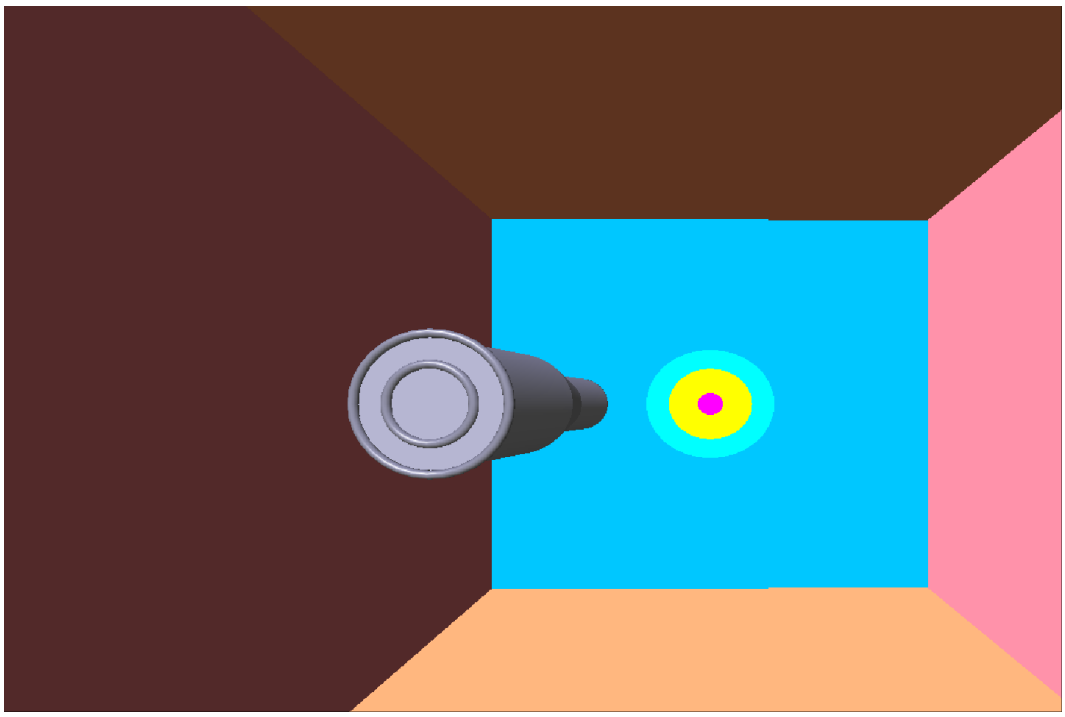

Figure 1: Before Firing the Shot

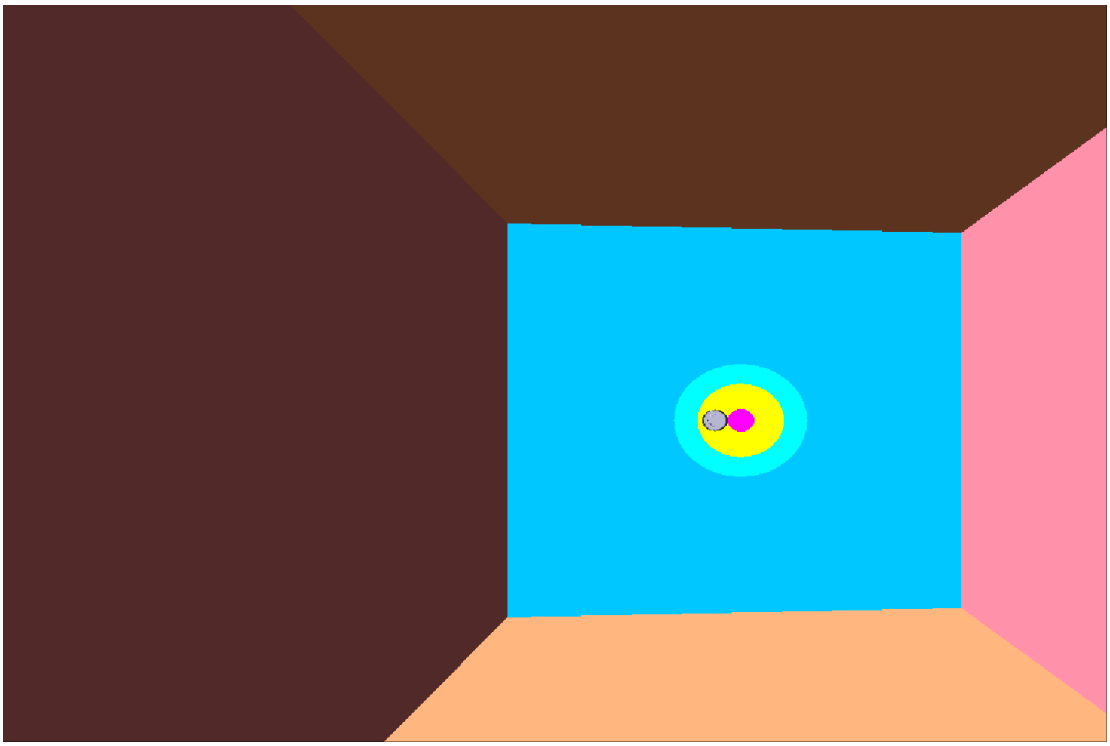

Figure 2: After Firing the Shot

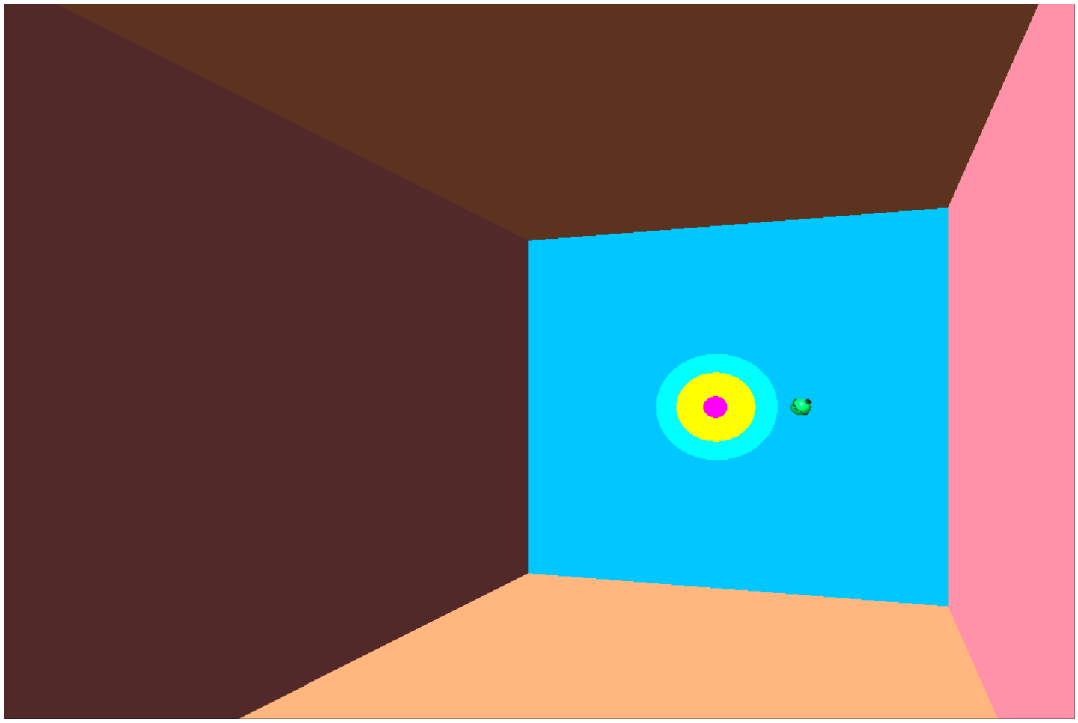

Figure 3: After Throwing the Bomb

The Code:

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include #include <stdio.h>
#include <math.h>
using namespace std; //for using std::out and similar features
#define PI 3.14159265 #define toRad 0.01745329251
```

```
//global structs typedef struct vector {
 float x;
 float y;
 float z;
 vector(float x, float y, float z) {
this->x = x; this->y = y; this->z =z; } vector() { this->x = 0; this->y = 0; this->z =0; } std::string
toString() { return "["+ std::to_string(x) + ", "+ std::to_string(y) + ", "+std::to_string(z)+"]" ; }
vector unitVector() { float magnitude = sqrtf(pow(x,2)+pow(y,2)+pow(z,2)); vector
unitVector(x/magnitude,y/magnitude,z/magnitude); return unitVector; } void set(float
newx, float newy, float newz) { x = newx; y = newy; z = newz; } void set(vector* vector) { x =
vector->x; y = vector->y; z = vector->z; } float magnitude() { return
sqrtf(pow(x,2)+pow(y,2)+pow(z,2)); } } vector; typedef struct quadraple { float a; float x; float
y; float z; quadraple(float a, float x, float y, float z) { this-> a = a; this->x = x; this->y = y; this-
>z =z; } quadraple() { this-> a = 0; this->x = 0; this->y = 0; this->z =0; } void set(float a, float x,
float y, float z) { this-> a = a; this->x = x; this->y = y; this->z =z; } void set(quadraple*
quadraple) { this-> a = quadraple->a; this->x = quadraple->x; this->y = quadraple->y; this->z
= quadraple->z; } std::string toString() { return "["+ std::to_string(a) + ", "+ std::to_string(x) +
", "+ std::to_string(y) + std::to_string(z)+"]" ; } } quadraple; typedef struct character { vector
*translation; quadraple *rotation; quadraple *deepRotation; vector
*firingInitialTranslation; quadraple *firingInitialRotation; float trajectoryXrotation; float
trajectoryYrotation; float initialTrajectoryYrotation; float bezierTranslation; bool isFiring;
bool hasFired; vector bezierTranslationPoints [4]; character(vector *translation, quadraple
*rotation, quadraple *deepRotation, vector *firingInitialTranslation, quadraple
*firingInitialRotation) { this->translation = translation; this->rotation = rotation; this-
>deepRotation = deepRotation; this->firingInitialTranslation = firingInitialTranslation; this-
>firingInitialRotation = firingInitialRotation; this->trajectoryXrotation = 0; this-
>trajectoryYrotation = 0; this->bezierTranslation = 0; this->isFiring = false; this->hasFired =
false; } void setTranslation(vector toTranslate) { translation->x = toTranslate.x; translation-
>y = toTranslate.y; translation->z = toTranslate.z; } void setRotation(quadraple toRotate)
{ rotation->a = toRotate.a; rotation->x = toRotate.x; rotation->y = toRotate.y; rotation->z =
toRotate.z; } void setRotation(float a, float x, float y, float z) { rotation->a = a; rotation->x = x;
rotation->y = y; rotation->z = z; } void resetAttrs() { this->bezierTranslation = 0; this->isFiring
= false; this->hasFired = false; } } character;

typedef struct scoreBoardTarget { vector *translation; double radius;
scoreBoardTarget(vector *translation, double radius) { this->translation = translation; this-
>radius = radius; } } scoreBoardTarget;
```

typedef struct gameStatus { std::string gameMode; int currCharacter; bool gameOver; bool inGameControls; bool isReplayMode; bool replaying; int replayCam; int score; gameStatus(std::string gameMode, int currCharacter) { this->gameMode = gameMode; this->currCharacter = currCharacter; this->gameOver = false; this->inGameControls = true; this->isReplayMode = false; this->replayCam = 1; this->score = 0; this->replaying = false; } void switchCharacter() { ++currCharacter%=3; } void toggleReplayCam() { ++replayCam%=2; } void reset() { inGameControls = true; gameOver = false; isReplayMode = false; replayCam = 1; this->replaying = false; } } gameStatus; typedef struct gameCamera { double eyeX; double eyeY; double eyeZ; double centerX; double centerY; double centerZ; double upX; double upY; double upZ; vector DefaultEye; vector DefaultCenter; vector DefaultUp; gameCamera(double eyeX, double eyeY, double eyeZ, double centerX, double centerY, double centerZ, double upX, double upY, double upZ) { this->eyeX = eyeX; this->eyeY = eyeY; this->eyeZ = eyeZ; this->centerX = centerX; this->centerY = centerY; this->centerZ = centerZ; this->upX = upX; this->upY = upY; this->upZ = upZ; vector DefaultEye(eyeX,eyeY,eyeZ); vector DefaultCenter(centerX,centerY,centerZ); vector DefaultUp(upX,upY,upZ); this->DefaultEye = DefaultEye; this->DefaultCenter = DefaultCenter; this->DefaultUp = DefaultUp; } void incrementEye(double eyeX, double eyeY, double eyeZ) { this->eyeX += eyeX; this->eyeY += eyeY; this->eyeZ += eyeZ; } void setEye(double eyeX, double eyeY, double eyeZ) { this->eyeX = eyeX; this->eyeY = eyeY; this->eyeZ = eyeZ; } void setCenter(double centerX, double centerY, double centerZ) { this->centerX = centerX; this->centerY = centerY; this->centerZ = centerZ; } void set(double eyeX, double eyeY, double eyeZ, double centerX, double centerY, double centerZ, double upX, double upY, double upZ) { this->eyeX = eyeX; this->eyeY = eyeY; this->eyeZ = eyeZ; this->centerX = centerX; this->centerY = centerY; this->centerZ = centerZ; this->upX = upX; this->upY = upY; this->upZ = upZ; } void reset(){ this->eyeX = DefaultEye.x; this->eyeY = DefaultEye.y; this->eyeZ = DefaultEye.z; this->centerX = DefaultCenter.x; this->centerY = DefaultCenter.y; this->centerZ = DefaultCenter.z; this->upX = DefaultUp.x; this->upY = DefaultUp.y; this->upZ = DefaultUp.z; } std::string eyeToString() { return "[" + std::to_string(eyeX) + ", " + std::to_string(eyeY) + ", " + std::to_string(eyeZ) + "]" ; } } gameCamera;

// Function signatures // Modeling void createBullet(); void createGrenade(); void createShurikenPart(); void createShuriken(); void createWall(); void createRoom(); void createTarget(); // Gameplay void initGame(); void endGame(); void drawGame(); bool hitWall(character* character); bool hitTarget(character* character); void calculateScore(character* character); // Motion int* bezier(float t, int* p0,int* p1,int* p2,int* p3); void changeCharacterTrajectoryAimLogic(int direction); void fireCharacter(); void fireBullet(character* bulletCharacter); void fireGrenade(character*

grenadeCharacter); void fireGrenadeStart(character* grenadeCharacter); void fireGrenadeLogic(character* grenadeCharacter); void fireShuriken(character* shurikenCharacter); void fireShurikenStart(character* grenadeCharacter); void fireShurikenLogic(character* grenadeCharacter); void characterHit(); // Replay void replay(); void replayFiringCamLogic(); // I/O void passM(int x,int y); void keyUp(unsigned char k, int x,int y); // Environment Configuration void setupCamera(); void setupLights(); // basic openGL void Display(); void Anim(); // END Function signatures

//global variables

static int windowHeight = 720; static int windowWidth = 1080; //double r=1; //int rd=1;

//Basic game state

gameStatus gameStat("basic", 0); //0 for bullet, 1 for grenade, 2 for shuriken gameCamera gameCam(0, 0, 180, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); // Defining default camera

vector mainCharacterTranslation(0,0,120); vector mainCharacterInitialTranslation(0,0,0); quadraple mainCharacterRotation(0,0,0,0); quadraple mainCharacterInitialRotation(0,0,0,0); quadraple mainCharacterDeepRotation(0,0,0,0); character mainCharacter(&mainCharacterTranslation, &mainCharacterRotation, &mainCharacterDeepRotation,&mainCharacterInitialTranslation ,&mainCharacterInitialRotation);

vector targetTranslation(0,0,5); scoreBoardTarget target(&targetTranslation, 20);

//GLuint texEarthID;

//trials

//end of initializations

void createBullet (character* thisCharacter) { glPushMatrix(); glRotated(thisCharacter->rotation->a ,thisCharacter->rotation->x, thisCharacter->rotation->y, thisCharacter->rotation->z); glRotated(thisCharacter->trajectoryXrotation ,1, 0,0); glTranslated(thisCharacter->translation->x, thisCharacter->translation->y, thisCharacter->translation->z); glRotated(thisCharacter->trajectoryYrotation ,0, 1,0);

glPushMatrix();
glRotated(thisCharacter->deepRotation->a ,thisCharacter->deepRotation->x, thisCharacter->deepRotation->y, thisCharacter->deepRotation->z);

```
glPushMatrix();
glColor3f(0.5, 0.5, 0.5);
GLUquadricObj * qobj4;
qobj4 = gluNewQuadric();
gluCylinder(qobj4, 0.2, 2, 3, 100,100);
glPopMatrix();

glPushMatrix();
glColor3f(0.5, 0.5, 0.5);
glTranslated(0, 0, 3);
GLUquadricObj * qobj3;
qobj3 = gluNewQuadric();
gluCylinder(qobj3, 2, 2, 5, 100,100);
glPopMatrix();


glPushMatrix();
glColor3f(0.5, 0.5, 0.5);
glTranslated(0, 0, 5+3);
GLUquadricObj * qobj2;
qobj2 = gluNewQuadric();
gluCylinder(qobj2, 2, 3, 4, 100,100);
glPopMatrix();



glPushMatrix();
glColor3f(0.5, 0.5, 0.5);
glTranslated(0, 0, 5+4+3);
GLUquadricObj * qobj;
qobj = gluNewQuadric();
gluCylinder(qobj, 3, 3, 7, 100,100);
glPopMatrix();


glPushMatrix();
glColor3f(0.5, 0.5, 0.5);
glTranslated(0, 0, 7+5+4+3);
```

```
gluDisk(qobj, 0.001, 3, 32, 32);
glPopMatrix();

glPushMatrix();
glTranslated(0, 0, 7+5+4+3);
glColor3f(0.4, 0.4, 0.4);
glutSolidTorus(0.2,3.2, 32, 32);
glPopMatrix();

glPushMatrix();
glTranslated(0, 0, 7+5+4+3+0.01);
glColor3f(0.4, 0.4, 0.4);
glutSolidTorus(0.2,1.8, 32, 32);
glPopMatrix();


glPopMatrix();
glPopMatrix();


}

void createGrenade (character* thisCharacter) { glPushMatrix(); glRotated(thisCharacter-
>rotation->a ,thisCharacter->rotation->x, thisCharacter->rotation->y, thisCharacter-
>rotation->z); glRotated(thisCharacter->trajectoryXrotation, 1, 0, 0);
glTranslated(thisCharacter->translation->x, thisCharacter->translation->y, thisCharacter-
>translation->z); GLUquadricObj * qobj; qobj = gluNewQuadric();

glPushMatrix();
glRotated(thisCharacter->deepRotation->a ,thisCharacter->deepRotation->x,
thisCharacter->deepRotation->y, thisCharacter->deepRotation->z);

glPushMatrix();
glRotated(90, 1, 0, 0); //REMOVE
glScaled(0.3, 0.3, 0.3); // scaling must be done here

glPushMatrix();
glColor3f(0,0.5,0.3);
glTranslated(0, 0, -12);
```

```
gluDisk(qobj, 0.001, 3, 32, 32);
glPopMatrix();

glPushMatrix();
glColor3f(0,0.4,0.2);
glTranslated(0, 0, -12);
gluCylinder(qobj, 3, 3, 3, 32,32);
glPopMatrix();

glPushMatrix();
glColor3f(0,0.4,0.2);
glutSolidTorus(1, 10, 32, 32);
glPopMatrix();

glPushMatrix();
glColor3f(0,0.8,0.3);
gluQuadricNormals(qobj, GLU_SMOOTH);
glutSolidSphere(10, 32, 32);
glPopMatrix();

glPopMatrix();


glPopMatrix();

glPopMatrix();


}

void createShurikenPart () { glPushMatrix();

glPushMatrix();
glBegin(GL_POLYGON);
glColor3f(1, 0.3, 1);
glVertex3f(5, 10, 0);
glVertex3f(20, 20, 0.0f);
glVertex3f(10,10, 3);
glEnd();
```

```
glPopMatrix();

glPushMatrix();
glBegin(GL_POLYGON);
glColor3f(0.7, 0.8, 0.3);
glVertex3f(10,10, 3);
glVertex3f(20, 20, 0);
glVertex3f(10, 5, 0.0f);
glEnd();
glPopMatrix();

glPushMatrix();// square shape matrix
glBegin(GL_POLYGON);
glColor3f(0.6, 0.7, 0.3);
glVertex3f(5, 5, 0);
glVertex3f(10, 5, 0.0f);
glVertex3f(10,10, 3);
glVertex3f(5,10,0);
glEnd();
glPopMatrix();
// bottom mirror
glPushMatrix();
glBegin(GL_POLYGON);
glColor3f(0.7, 0.3, 0.8);
glVertex3f(5, 10, 0);
glVertex3f(20, 20, 0.0f);
glVertex3f(10,10, -3);
glEnd();
glPopMatrix();

glPushMatrix();
glBegin(GL_POLYGON);
glColor3f(0.7, 1, 0.8);
glVertex3f(10,10, -3);
glVertex3f(20, 20, 0);
glVertex3f(10, 5, 0.0f);
glEnd();
glPopMatrix();
```

```
glPushMatrix();// square shape matrix
glBegin(GL_POLYGON);
glColor3f(0.6, 1, 0.3);
glVertex3f(5, 5, 0);
glVertex3f(10, 5, 0.0f);
glVertex3f(10,10, -3);
glVertex3f(5,10,0);
glEnd();
glPopMatrix();


glPopMatrix();


}

void createShuriken(character* thisCharacter) { glPushMatrix(); glRotated(thisCharacter-
>trajectoryXrotation, 1, 0, 0); glRotated(thisCharacter->rotation->a ,thisCharacter-
>rotation->x, thisCharacter->rotation->y, thisCharacter->rotation->z);
glTranslated(thisCharacter->translation->x, thisCharacter->translation->y, thisCharacter-
>translation->z); glRotated(thisCharacter->trajectoryYrotation, 0, 1, 0); glPushMatrix();
glRotated(thisCharacter->deepRotation->a ,thisCharacter->deepRotation->x,
thisCharacter->deepRotation->y, thisCharacter->deepRotation->z);

glPushMatrix();
glRotated(90, 1, 0, 0);
glScaled(0.3, 0.3, 0.3);

glPushMatrix();
createShurikenPart();
glPopMatrix();

glPushMatrix();
glRotated(240, 0, 0, 1);
createShurikenPart();
glPopMatrix();

glPushMatrix();
```

```
glRotated(120, 0, 0, 1);
createShurikenPart();
glPopMatrix();


glPushMatrix();
glColor3f(0.4,0.4,0.4);
glutSolidTorus(2, 5, 32, 100);
glPopMatrix();

glPushMatrix();
glColor3f(1,0,0);
glRotated(60, 1, 1, 0);
glutSolidCube(5);
glPopMatrix();
glPopMatrix();

glPopMatrix();

glPopMatrix();


}

void createWall () { glPushMatrix(); glBegin(GL_POLYGON); glVertex3f(-70, -70, 0);
glVertex3f(70, -70, 0.0f); glVertex3f(70,70, 0); glVertex3f(-70,70, 0); glEnd(); glPopMatrix(); }

void createRoom () { //face glPushMatrix(); glNormal3f(0.6, 0.4, 0.7); glColor3f(0,0.6,1);
glTranslated(0, 0, 0); createWall(); glPopMatrix();

glPushMatrix();
glNormal3f(-1, 0, 0);
glColor3f(0.8,0.4,0.4);
glTranslated(70, 0, 70);
glRotated(90, 0, 1, 0);
createWall();
glPopMatrix();

glPushMatrix();
```

```
glNormal3i(1, 0, 0);
glColor3f(0.8,0.4,0.4);
glTranslated(-70, 0, 70);
glRotated(90, 0, 1, 0);
createWall();
glPopMatrix();

//ceiling
glPushMatrix();
glNormal3f(0, -1, 0);
glColor3f(0.9,0.5,0.3);
glTranslated(0, 70, 70);
glRotated(90, 1, 0, 0);
createWall();
glPopMatrix();

//floor
glPushMatrix();
glNormal3f(0, 1, 0);
glColor3f(0.9,0.5,0.3);
glTranslated(0, -70, 70);
glRotated(90, 1, 0, 0);
createWall();
glPopMatrix();


}

void createTarget (scoreBoardTarget* target) { GLUquadricObj * qobj; qobj =
gluNewQuadric(); glPushMatrix(); glTranslated(target->translation->x, target->translation-
>y, target->translation->z);

glPushMatrix();
glTranslated(0, 0, 0);

glPushMatrix();
glColor3f(1, 0, 1);
gluDisk(qobj, 0.001, 4, 32, 32);
glPopMatrix();
```

```
glPushMatrix();
glColor3f(1, 1, 0);
gluDisk(qobj, 4, 13, 32, 32);
glPopMatrix();

glPushMatrix();
glColor3f(0, 1, 1);
gluDisk(qobj, 13, 20, 32, 32);
glPopMatrix();

glPopMatrix();

glPopMatrix();


}

void drawGame(scoreBoardTarget* target ,character* character) { createRoom();
createTarget(target); switch(gameStat.currCharacter) { case 0: createBullet(character);
break; case 1: createGrenade(character); break; case 2: createShuriken(character);
break; } }

//Motion Logic

//Aiming

void changeCharacterTrajectoryAimLogic(int direction) { bool canIncreaseY = true; bool
canDecreaseY = true; switch (gameStat.currCharacter) { case 0:
if(mainCharacter.rotation->a >= 24) canIncreaseY = false; if(mainCharacter.rotation->a <=
-24) canDecreaseY = false; break; case 1: if(mainCharacter.rotation->a >= 28)
canIncreaseY = false; if(mainCharacter.rotation->a <= -28) canDecreaseY = false; break;
case 2: if(mainCharacter.rotation->a >= 26) canIncreaseY = false;
if(mainCharacter.rotation->a <= -26) canDecreaseY = false; break; } if(direction == 1 &&
canDecreaseY) mainCharacter.setRotation(mainCharacter.rotation->a-2, 0,1,0);
if(direction == 2 && canIncreaseY) mainCharacter.setRotation(mainCharacter.rotation-
>a+2, 0,1,0); if(direction == 3 && mainCharacter.trajectoryXrotation>=-
10){ mainCharacter.trajectoryXrotation-=2; } if(direction == 4 &&
mainCharacter.trajectoryXrotation<=10){ mainCharacter.trajectoryXrotation+=2; }
```

```
}

// Firing

void fireCharacter() { // Save initial firing values for replay later.
if(!mainCharacter.isFiring){ mainCharacter.firingInitialTranslation-
>set(mainCharacter.translation); mainCharacter.firingInitialRotation-
>set(mainCharacter.rotation); mainCharacter.initialTrajectoryYrotation =
mainCharacter.trajectoryYrotation; } // Call character firing function based on the current
active character switch(gameStat.currCharacter) { case 0: fireBullet(&mainCharacter);
break; case 1: fireGrenade(&mainCharacter); break; case 2:
fireShuriken(&mainCharacter); break; } if(gameStat.isReplayMode &&
gameStat.replaying){ replayFiringCamLogic(); } }

void replayFiringCamLogic(){ if (mainCharacter.isFiring) { switch (gameStat.replayCam)
{ case 0: switch (gameStat.currCharacter) { case 0:
gameCam.setCenter(mainCharacter.translation->x, mainCharacter.translation->y,
mainCharacter.translation->z); break; case 1:
gameCam.setCenter(mainCharacter.translation->x, mainCharacter.translation-
>y>0?mainCharacter.translation->y-12:mainCharacter.translation->y+25,
mainCharacter.translation->z>0?mainCharacter.translation->z-
10:mainCharacter.translation->z+10); break; case 2: if(mainCharacter.translation-
>z>0){ gameCam.setEye(mainCharacter.translation->x-15, mainCharacter.translation->y,
mainCharacter.translation->z); } gameCam.setCenter(mainCharacter.translation-
>x>0?mainCharacter.translation->x-10:mainCharacter.translation->x+10,
mainCharacter.translation->y, mainCharacter.translation->z); break; } break; case 1:
switch (gameStat.currCharacter) { case 0:
gameCam.setCenter(mainCharacter.translation->x, mainCharacter.translation->y,
mainCharacter.translation->z-40); gameCam.setEye(mainCharacter.translation->x,
mainCharacter.translation->y, mainCharacter.translation->z-10); break; case 1:
gameCam.setEye(mainCharacter.translation->x, mainCharacter.translation->y,
mainCharacter.translation->z+20); gameCam.setCenter(mainCharacter.translation->x,
mainCharacter.translation->y>0?mainCharacter.translation->y-
12:mainCharacter.translation->y+25, mainCharacter.translation-
>z>0?mainCharacter.translation->z-10:mainCharacter.translation->z+10); break; case 2:
gameCam.setEye(mainCharacter.translation->x-15, mainCharacter.translation->y,
mainCharacter.translation->z); gameCam.setCenter(mainCharacter.translation-
>x>0?mainCharacter.translation->x-10:mainCharacter.translation->x+10,
mainCharacter.translation->y, mainCharacter.translation->z-70);
```

```
        }
         break;
    }
}else{
    gameCam.reset();

}


}
```

```
void fireBullet(character* bulletCharacter) { bulletCharacter->isFiring = true; // Rotation of bullet around its axis bulletCharacter->deepRotation->a +=2; bulletCharacter->deepRotation->z= 1; // Z axis default translation bulletCharacter->translation->z -= 1*0.5; // X axis conditional translation if (bulletCharacter->rotation->a != 0) { if(bulletCharacter->rotation->a < 0) { float slope = tan ((180+bulletCharacter->rotation->a) * toRad); bulletCharacter->translation->x += slope*0.01; } else { float slope = tan ((180-bulletCharacter->rotation->a) * toRad); bulletCharacter->translation->x -= slope*0.01; } } if (bulletCharacter->trajectoryYrotation != 0) { if(bulletCharacter->trajectoryYrotation < 0) { float slope = tan ((180+bulletCharacter->trajectoryYrotation) * toRad); bulletCharacter->translation->x -= slope*0.5; } else { float slope = tan ((180-bulletCharacter->trajectoryYrotation) * toRad); bulletCharacter->translation->x += slope*0.5; } } // hit or miss logic if(hitTarget(bulletCharacter) || hitWall(bulletCharacter)) { calculateScore(bulletCharacter); bulletCharacter->isFiring = false; characterHit(); } }

void calculateScore(character* character){ if(hitTarget(character) && !gameStat.isReplayMode){ cout << "score: " << ++gameStat.score << "\n"; } }

void fireGrenade(character* grenadeCharacter) { if(!grenadeCharacter->isFiring) { fireGrenadeStart(grenadeCharacter); } else { fireGrenadeLogic(grenadeCharacter); } }

void fireGrenadeStart(character* grenadeCharacter) { float endX = 0; if(grenadeCharacter->trajectoryYrotation < 0) { float slope = tan((180+grenadeCharacter->trajectoryYrotation) * toRad); endX -= slope*grenadeCharacter->translation->z; } if(grenadeCharacter->trajectoryYrotation > 0) { float slope = tan((180-grenadeCharacter->trajectoryYrotation) * toRad); endX += slope*grenadeCharacter->translation->z; } grenadeCharacter->bezierTranslationPoints[0] = vector(grenadeCharacter->translation->x,grenadeCharacter->translation->y,grenadeCharacter->translation->z); grenadeCharacter->bezierTranslationPoints[1] = vector(0,grenadeCharacter->translation-
```

```
>y+65,grenadeCharacter->translation->z); grenadeCharacter->bezierTranslationPoints[2]
= vector(endX,grenadeCharacter->translation->y+65,0); grenadeCharacter-
>bezierTranslationPoints[3] = vector(endX,-45,0); // -60 in Y is floor, since floor is at -70 and
radius of grenade is 10 grenadeCharacter->isFiring = true; }

void fireGrenadeLogic(character* grenadeCharacter) { int p0[2] =
{static_cast(grenadeCharacter->bezierTranslationPoints[0].z),
static_cast(grenadeCharacter->bezierTranslationPoints[0].y)}; int p1[2] =
{static_cast(grenadeCharacter->bezierTranslationPoints[1].z),
static_cast(grenadeCharacter->bezierTranslationPoints[1].y)}; int p2[2] =
{static_cast(grenadeCharacter->bezierTranslationPoints[2].z),
static_cast(grenadeCharacter->bezierTranslationPoints[2].y)}; int p3[2] =
{static_cast(grenadeCharacter->bezierTranslationPoints[3].z),
static_cast(grenadeCharacter->bezierTranslationPoints[3].y)};

if (!(grenadeCharacter->bezierTranslation>1)) {
  grenadeCharacter->deepRotation->a+=10;
  grenadeCharacter->deepRotation->z=1;
  grenadeCharacter->deepRotation->y=1;
  grenadeCharacter->bezierTranslation+=0.03;
  int* p = bezier(grenadeCharacter->bezierTranslation,p0,p1,p2,p3);
  grenadeCharacter->translation->z = p[0];
  grenadeCharacter->translation->y = p[1];
  if(grenadeCharacter->trajectoryYrotation > 0) {
    float slope = tan((180+grenadeCharacter->trajectoryYrotation) * toRad);
    grenadeCharacter->translation->x+=slope*0.8;
  } else if(grenadeCharacter->trajectoryYrotation < 0) {
    float slope = tan((180-grenadeCharacter->trajectoryYrotation) * toRad);
    grenadeCharacter->translation->x-=slope*0.8;
  } else {
    grenadeCharacter->translation->z+=1;
  }
}
if (hitTarget(grenadeCharacter) || hitWall(grenadeCharacter)) {
  calculateScore(grenadeCharacter);
  grenadeCharacter->isFiring = false;
  characterHit();
```

```
}


}

void fireShurikenLogic(character* shurikenCharacter) { int p0[2] =
{static_cast(shurikenCharacter->bezierTranslationPoints[0].z),
static_cast(shurikenCharacter->bezierTranslationPoints[0].x)}; int p1[2] =
{static_cast(shurikenCharacter->bezierTranslationPoints[1].z),
static_cast(shurikenCharacter->bezierTranslationPoints[1].x)}; int p2[2] =
{static_cast(shurikenCharacter->bezierTranslationPoints[2].z),
static_cast(shurikenCharacter->bezierTranslationPoints[2].x)}; int p3[2] =
{static_cast(shurikenCharacter->bezierTranslationPoints[3].z),
static_cast(shurikenCharacter->bezierTranslationPoints[3].x)};

if (!(shurikenCharacter->bezierTranslation>1)) {
   shurikenCharacter->deepRotation->a+=20;
   shurikenCharacter->deepRotation->y=1;
   shurikenCharacter->bezierTranslation+=0.01;
   int* p = bezier(shurikenCharacter->bezierTranslation,p0,p1,p2,p3);
   shurikenCharacter->translation->z = p[0];
   shurikenCharacter->translation->x = p[1];
}
if (hitTarget(shurikenCharacter) || hitWall(shurikenCharacter)) {
   shurikenCharacter->isFiring = false;
   calculateScore(shurikenCharacter);
   characterHit();
}


}

void fireShurikenStart(character* shurikenCharacter) { float endX; if (shurikenCharacter-
>trajectoryYrotation<0) endX = shurikenCharacter->translation->ztan(shurikenCharacter-
>trajectoryYrotationtoRad)-30; else if (shurikenCharacter->trajectoryYrotation>0) endX =
shurikenCharacter->translation->ztan(shurikenCharacter->trajectoryYrotationtoRad)-60;
else endX = shurikenCharacter->translation->ztan(shurikenCharacter->rotation->atoRad)-
70;
```

```
shurikenCharacter->bezierTranslationPoints[0] = vector(shurikenCharacter->translation->x,shurikenCharacter->translation->y,shurikenCharacter->translation->z);
shurikenCharacter->bezierTranslationPoints[1] = vector(shurikenCharacter->translation->z*tan(shurikenCharacter->rotation->a*toRad)+40,0,shurikenCharacter->translation->z-90);
shurikenCharacter->bezierTranslationPoints[2] = vector(shurikenCharacter->translation->z*tan(shurikenCharacter->rotation->a*toRad)+40,0,shurikenCharacter->translation->z-90);
shurikenCharacter->bezierTranslationPoints[3] = vector(endX,0,-60); // -60 in Y is floor
shurikenCharacter->isFiring = true;


}

void fireShuriken(character* shurikenCharacter) { if(!shurikenCharacter->isFiring) { fireShurikenStart(shurikenCharacter); } else { fireShurikenLogic(shurikenCharacter); } }

bool hitWall(character* character){ switch (gameStat.currCharacter) { case 0: return (character->translation->z<=3 || character->translation->x<-60 || character->translation->x>60 || character->translation->y<-60 || character->translation->y>60); break; case 1: return (character->translation->z<=10 || character->translation->x<-60 || character->translation->x>60 || character->translation->y<-60 || character->translation->y>60); break; case 2: return (character->translation->z<=12 || character->translation->x<-40 || character->translation->x>40 || character->translation->y<-60 || character->translation->y>60); break; } return false; }

bool hitTarget(character* character) { switch (gameStat.currCharacter) { case 0: return character->translation->z == target.translation->z && character->translation->x > target.translation->x - target.radius && character->translation->x < target.translation->x + target.radius; case 1: return character->translation->z <= target.translation->z+5 && character->translation->z >= target.translation->z && character->translation->y > target.translation->y - target.radius && character->translation->y < target.translation->y + target.radius && character->translation->x > target.translation->x - target.radius && character->translation->x < target.translation->x + target.radius; break; case 2: return character->translation->z <= target.translation->z+10 && character->translation->z >= target.translation->z && character->translation->y > target.translation->y - target.radius && character->translation->y < target.translation->y + target.radius && character->translation->x > target.translation->x - target.radius-3 && character->translation->x < target.translation->x + target.radius+3; break; } return false; }
```

```
// Bezier curve calculation int* bezier(float t, int* p0, int* p1, int* p2, int* p3) { static int
res[2]; // Make static to avoid returning address of local variable res[0] = pow((1-
t),3)p0[0]+3tpow((1-t),2)p1[0]+3pow(t,2)(1-t)p2[0]+pow(t,3)p3[0]; res[1] = pow((1-
t),3)p0[1]+3tpow((1-t),2)p1[1]+3pow(t,2)(1-t)*p2[1]+pow(t,3)*p3[1]; return res; }

void characterHit() { gameStat.inGameControls = false; gameStat.isReplayMode = true; //
Use nanosleep instead of for loop for delay struct timespec ts; ts.tv_sec = 0; ts.tv_nsec =
100000000; // 100ms delay nanosleep(&ts, NULL);

if(!gameStat.replaying){
  replay();
}


}

void replay(){ if(!gameStat.inGameControls){ mainCharacter.resetAttrs();
mainCharacter.translation->set(mainCharacter.firingInitialTranslation);
mainCharacter.rotation->set(mainCharacter.firingInitialRotation);
mainCharacter.deepRotation->set(0,0,0,0); gameStat.replaying = true;
mainCharacter.trajectoryYrotation = mainCharacter.initialTrajectoryYrotation;
fireCharacter(); } }

void initGame() { gameCam.reset(); gameStat.reset(); mainCharacter.resetAttrs();
mainCharacter.trajectoryYrotation = 0; mainCharacter.translation->set(0,0,140);
mainCharacter.rotation->set(0,0,0,0); mainCharacter.deepRotation->set(0,0,0,0);
mainCharacter.trajectoryXrotation = 0; target.translation->set(0,0,5); }

void endGame() { gameStat.gameOver = true; exit(0); }

//OPENGL FUNCS

void Display() { glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); glPushMatrix();

if(!gameStat.gameOver) {
  setupCamera();
  setupLights();
  drawGame(&target, &mainCharacter);
}
```

```
glPopMatrix();
glutSwapBuffers(); // Changed from glFlush() for double buffering


}

void anim() { if(mainCharacter.isFiring) { fireCharacter(); } glutPostRedisplay(); }

void passM(int x,int y) { float mappedX = (x - (windowWidth/2)); if(mappedX>-
windowWidth/4.2 && mappedX < windowWidth/4.2) gameCam.eyeX = mappedX0.2;
if(!mainCharacter.isFiring && !gameStat.isReplayMode){ if (mappedX*0.4<80 &&
mappedX*0.4 >-80) { mainCharacter.trajectoryYrotation = mappedX0.4; } } }

void keyUp(unsigned char k, int x, int y) { if(gameStat.inGameControls){ switch (k) { case
'w': target.translation->z++; break; case 's': if(target.translation->z>=2) { target.translation-
>z--; } break; case 'd': if(target.translation->x!=(45)) { target.translation->x++; } break; case
'a': if(target.translation->x!=(-45)) { target.translation->x--; } break; case 32: // space bar
if(!mainCharacter.hasFired) fireCharacter(); break; case 48: // '0' key
gameStat.switchCharacter(); break; case 49: // '1' key case 'j':
changeCharacterTrajectoryAimLogic(1); break; case 50: // '2' key case 'l':
changeCharacterTrajectoryAimLogic(2); break; case 'i':
changeCharacterTrajectoryAimLogic(3); break; case 'k':
changeCharacterTrajectoryAimLogic(4); break; } } switch (k) { case 27: // ESC key
endGame(); break; case 'n': initGame(); break; case 'r': replay(); break; case 9: // Tab key
gameStat.toggleReplayCam(); break; } glutPostRedisplay(); }

void setupCamera() { glMatrixMode(GL_PROJECTION); glLoadIdentity(); glOrtho(1, 1, -1, 1, -
1, 1); gluPerspective(80, static_cast(windowWidth) / windowHeight, 0.001, 1000);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(gameCam.eyeX, gameCam.eyeY, gameCam.eyeZ, gameCam.centerX,
gameCam.centerY, gameCam.centerZ, gameCam.upX, gameCam.upY, gameCam.upZ);


}

void setupLights() { GLfloat ambient[] = { 0.7f, 0.7f, 0.7f, 1.0f }; GLfloat diffuse[] = { 0.6f, 0.6f,
0.6f, 1.0f }; GLfloat specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; GLfloat shininess[] = { 100.0f };
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient); glMaterialfv(GL_FRONT,
```

```
GL_DIFFUSE, diffuse); glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
glMaterialfv(GL_FRONT, GL_SHININESS, shininess);

GLfloat lightIntensity[] = { 0.7f, 0.7f, 1.0f, 1.5f };
GLfloat lightPosition[] = { 0.0f, 60.0f, 70.0f, 0.0f };
GLfloat lightDirection[] = { 0.0f, -1.0f, -1.0f };
glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightIntensity);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, lightDirection);

GLfloat l1Diffuse[] = { 0.5f, 0.5f, 0.5f, 0.3f };
GLfloat l1Ambient[] = { 0.2f, 0.2f, 0.2f, 0.2f };
GLfloat l1Position[] = { 0.0f, 0.0f, 140.0f, 0.0f };
GLfloat l1Direction[] = { 0.0f, 0.0f, -1.0f };
glLightfv(GL_LIGHT1, GL_DIFFUSE, l1Diffuse);
glLightfv(GL_LIGHT1, GL_AMBIENT, l1Ambient);
glLightfv(GL_LIGHT1, GL_POSITION, l1Position);
glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, l1Direction);
glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 30.0f);
glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 90.0f);


}

int main(int argc, char** argv) { glutInit(&argc, argv); glutInitDisplayMode(GLUT_DOUBLE |
GLUT_RGB | GLUT_DEPTH);

glutInitWindowSize(windowWidth, windowHeight);
glutCreateWindow("Aim and Shoot");

// Register callbacks first
glutDisplayFunc(Display);
glutIdleFunc(anim);
glutPassiveMotionFunc(passM);
glutKeyboardUpFunc(keyUp);

// Then try to enter game mode
char gameModeString[24];
```

```
sprintf(gameModeString, "%dx%d:32", glutGet(GLUT_SCREEN_WIDTH),
glutGet(GLUT_SCREEN_HEIGHT));
glutGameModeString(gameModeString);

if (glutGameModeGet(GLUT_GAME_MODE_POSSIBLE)) {
   glutEnterGameMode();
   // Re-register callbacks after entering game mode
   glutDisplayFunc(Display);
   glutIdleFunc(anim);
   glutPassiveMotionFunc(passM);
   glutKeyboardUpFunc(keyUp);
}

glutSetCursor(GLUT_CURSOR_NONE);

glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
glViewport(0, 0, windowWidth, windowHeight);

glEnable(GL_DEPTH_TEST);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
glEnable(GL_NORMALIZE);
glEnable(GL_COLOR_MATERIAL);

glShadeModel(GL_SMOOTH);
glEnable(GL_TEXTURE_2D);

initGame();

glutMainLoop();
return 0;


}
```