Programming Assignment 1: Death Fireball Zombie Terror from Planet 9

Handed out: Tue, Sep 13. Due: Wed, Sep 28, 11:59:59pm. Late policy: up to 6 hours late: 5% of the total; up to 24 hours late: 10%, and then 20% for each additional 24 hours.

Overview: The goal of this assignment is to learn the basics of OpenGL and GLUT and two-dimensional geometry. You are to implement a simple 2-dimensional computer game. You have some flexibility in how you implement the game (e.g., changing the user interface or modifying the game’s behavior), subject to the requirement that your program contain all the essential elements outlined below.

The game takes place inside a 2-dimensional window. Along the top edge of the window is a water cannon. The can position, aim, and shoot the cannon, which sprays a jet of water. From the bottom of the window, a sequence of fireballs float upwards. The objective of the player is to extinguish each of the fireballs with the water cannon before it reaches the top of the window.

Fireball: Each fireball can be rendered as a colored, circular disk (which you can approximate by a many-sided polygon, that essentially looks circular). Once generated, a fireball rises up at a fixed velocity. A fireball has an initial radius, but whenever it is hit by the water cannon, the radius decreases by a constant amount. When the radius falls below a certain minimum value, the fireball vanishes.

The player does not want to allow any fireball to reach the top of the window. (This can be handled in various ways. For example, when the first fireball hits the top of the window, the player loses. Alternately, the game can maintain a score, and the player’s score could increase as fireballs are extinguished and decreased as balls reach the top of the window. You can implement this as you like.)

Cannon: The cannon sits at the top edge of the window. It can be rendered as a rectangle (or any other suggestive shape.) The user must be able to adjust both the cannon’s position and direction. (One easy way to handle this is to use the ‘a’ and ‘d’ keys to move the cannon left and right, respectively, and have the cannon point to whatever position the mouse is at.)

When the left mouse button is pressed down, the cannon fires a stream of water, until the button is released. The water stream can be implemented as a collection water particles, called droplets, which emerge from the cannon at a very high rate (e.g., one per update cycle). There are two ways that a water droplet ceases to exist. First, it can hit a fireball. Second, it can move outside the window. You may assume that water droplets move at a constant velocity, or, (for extra credit) you try something fancier, such as simulating gravity.

Smooth animation: I would recommend using glutIdleFunc() to continuously update the state of your game. You can track the elapsed time between updates using ftime (or whatever timing function your system provides). See the class web page for further information.

Resetting and Quitting: It should be possible to reset the game to its starting point, say by hitting the ‘r’ key and to quit the game, say by hitting the ‘q’ or ESC keys. For extra credit, you may also implement a capability to pause and resume the game, say, by hitting the ‘p’ key.

Resizing the Window: Your program should allow the user to resize the game window. When the window is resized, the game should be reset to a natural starting configuration. Resizing should
cause everything to scale accordingly. For example, if the window size is doubled, then all the 
graphics within the window should be doubled in size. Also, velocities should scale, so that if it 
takes $t$ time units for a fireball to rise from the bottom of the window to the top, then it should 
still take $t$ time units after resizing. Resizing should not cause the graphics to be distorted. (That 
is, circular balls should remain circular, and not be elongated into ellipses).

**Playability:** Assuming a reasonable window size, the game-play should be reasonable. That is, the 
fireballs should not be so numerous nor move so fast nor be so hard to extinguish that the game 
cannot be won. On the other hand, it should not be too easy either.

**Final Submission:** Submissions will be made through the Department’s submit server, https://submit.cs.umd.edu/. 
(The submit server is used only for uploading. All testing will be done by the TA.) Your submission 
will be in the form of a file archive. (You may use any standard archiving software, such as Winzip, 
WinRAR, or Unix tar and gzip. If you are unsure, check to see that the TA has your favorite archiver.) 
The submission should contain everything that the TA will need to compile, execute, and test your 
program. This will consist of:

- **Readme:** A file (e.g., *Readme.txt*), which explains everything the grader will need to know about 
  how to compile and run your program. For example, this will include the platform on which your 
  program runs (e.g., “Linux using g++” or “Windows using Visual Studio 2010”), how to compile 
  your program (very important), how to run and execute your program, any special features you 
  have implemented (very important), and any bugs or limitations that you are aware of. If you 
  borrowed code from elsewhere, even if you modified it, please mention the source here briefly. 
  If you are using MacOS with XCode (which the TA does not have), be sure that you provide 
  directions for compiling and running your program from a regular Unix-like command window.

- **Makefile or Solution files:** Include any files or instructions needed for compiling your program. 
  (E.g. a *Makefile* if you are on a Unix system or the *.sln* and *.vcproj* files for Visual Studio.

- **Source files:** Your program source files.

- **Resources:** Any additional files needed for execution (e.g., images or model files used by your pro-
  gram).

- **Omit:** Omit (especially large) binary files that are generated in the compilation process. This includes 
  executable files and object files. Excluding resources, if your final submission is bigger than 
  100Kb, you are probably including something unnecessary.

**Trial Submission:** Because we are using many different platforms, I would recommend that you perform 
a test submission of your project at least three days before the final deadline. (This is especially true 
for Mac users.) I will ask the TA to compile early submissions, and get back to you if he experiences 
any issues. Your trial submission does not need to do anything interesting. For example, it could just 
compile correctly and bring up a blank graphics window when executed.

**Programming Style:** We will be reading your code to see that you implemented everything in a reasonable 
manner. Although style does not constitute a major part of the final grade, we will deduct points for 
programs that are poorly documented or that have convoluted structure. Since many of you are not 
familiar with C++ programming, we will not deduct points for poor C++ programming style. But, try 
to do your best.
Optional Elements: Here are some ideas for extensions to your project for extra credit. (See the course syllabus on how extra credit points are counted.)

Improved Graphics: Replace our simple circular fireball and rectangular cannon with more interesting geometric shapes, with more interesting colors.

Multiple levels: Offer different levels, where the game behavior is different. (Rather than forcing the TA to play through each level, provide a user input that allows him to jump to a particular level.)

Start-up screen: Show a start-up screen when the game starts and an ending screen (e.g., showing the final score or the player’s results) when it exits.

Special Effects: Introduce any additional special effects that you like. For example, when a water droplet hits a fireball, in addition to shrinking the fireball’s motion may be deflected.

Tips:

Units: Decide which units you want to use in representing your world. For simple 2-dimensional projects list this, it is natural to simply use actual pixels as the unit of distance, but this is not the best idea, because rescaling the game forces you to explicitly scale all the object sizes and shapes. What I did was to define a separate game rectangle, where all the action and drawing take place, and then use glOrtho2D to map this to the actual window. The height of my game rectangle is fixed to some convenient value (e.g., 1 or 100) and all my graphics are defined relative to this size. To avoid distortion, I defined the width of my game rectangle to be $w/h$, where $w$ is the graphics window width and $h$ is the graphics window height. By doing this, no scaling of sizes or velocities is needed.

State: All moving objects are characterized by their current physical state. This consists of the position $p$ of the center of the object and the velocity $v$ of the object. Both of these are vector quantities (the position is a point in space and the velocity is a vector). In general, you need to store all the information in order to render each object, to updates its position, and to detect and process any collision events. The naturally leads to an object-oriented approach, where you represent each object as a class, where the class variables store the object’s current state, and methods process the object’s possible actions and interactions.

Updates: With each update cycle (e.g., glut idle event), update the state. Based on the amount of time $\Delta t$ that has elapsed since the last update, the object position can be updated as $p \leftarrow p + \Delta t \cdot v$. After updating positions, check for collisions. (If objects are moving super fast, it is theoretically possible for one object to pass through another without detecting a collision. Don’t worry about this, since our objects will not be moving that fast.)

Collision Detection: To determine whether two circular objects collide, check whether the distance between their centers is smaller than the sum of their radii.
Homework 1: OpenGL and Geometry

Handed out Tuesday, Oct 4. Due at the start of class Tuesday, Oct 11. Late homeworks are not accepted, so turn in whatever you have done.

Problem 1. For each of the following geometric tasks, explain how you would solve it using the methods of affine and Euclidean geometry that we have discussed in class. (As much as possible, try to express your answers in terms of high-level operations, such as affine combinations, dot-product, and cross-product, rather than low-level coordinate manipulations or trigonometric functions.)

(a) Given a triangle $\triangle pqr$ in the plane, explain how to compute a fourth point $s$ such that $\{p, s, q, r\}$ defines a parallelogram which has $pq$ as a diagonal (see Fig. 1(a)).

(b) A viewer is located at a point $e$ in 3-dimensional space, and he sees a triangle defined by vertices $p$, $q$, and $r$. The front side of this triangle is the side from which $p$, $q$, and $r$ appear in counterclockwise order, and other side is called the back side. Assuming that $e$ is not coplanar with the triangle, determine whether the viewer sees the front side or back side of the triangle (see Fig. 1(b)).

(c) Four points $\{p, q, r, s\}$ define a rectangle in 3-dimensional space. Show how to compute a point $t$ such that a line segment from $t$ perpendicular to the rectangle hits the midpoint of the rectangle, is 2 units long, and is on the side such that vertices $\{p, q, r, s\}$ appear in counter clockwise order with respect to a viewer at $t$ (see Fig. 1(c)).

(d) Given a ball in 3-dimensional space whose center is at point $c$ and whose radius $r$, and given a line segment $pq$, does the segment intersect the ball (see Fig. 1(d))? (Hint: The easiest solution to this problem does not require knowing the equation of a sphere in 3-space, but I’ll give it to you anyway. A point $(x, y, z)$ lies within the ball if $(x - cx)^2 + (y - cy)^2 + (z - c)^2 \leq r^2$.)

![Figure 1: Problem 1 parts (a)–(d) and the Challenge Problem (e).](image)

Problem 2. You have a drawing rectangle of width $w$ and height $h$ (see Fig. 2(a)), which is to be mapped to a window of width $W$ and height $H$. The drawing rectangle is to be scaled uniformly so it fits entirely within the window, and (depending on the relationship of the two aspect ratios) either the
width matches the width of the window or the height matches the height of the drawing window. In either case, it is centered within the window (see Fig. 2(b) and (c)). Given the glViewport command needed to map the graphics to the desired window. Explain how you derived your answer.

![Figure 2: Problem 2.](image)

**Problem 3.** The following problem takes place in the $x, y$-plane. You are given two procedures (see the figure below):

- **drawFace()**: Draws the face of a clock. The face is inside a circle of radius 1, centered at the origin.
- **drawHand()**: Draws a hand of the clock. The length of the hand is 1, and it points straight up from the origin.

Using these two procedures, implement in OpenGL a drawing procedure **drawClock($x, y, r, hh, mm$)**. This draws a clock centered at the point $(x, y)$, of radius $r$, with hands adjusted so that it shows the time $hh : mm$. The parameter $hh$ is an integer from 0 to 11, indicating the hour, and the parameter $mm$ is an integer from 0 to 59, indicating the minute.

The hands should be drawn at the appropriate locations. (You may either arrange the short hand to coincide exactly with the hour or to place it proportionately between two hours, depending on the minute value.) The long hand’s length should be $3r/4$ and the short hand’s length should be $r/2$. On return, the Modelview matrix stack should be unchanged.

**Challenge Problem:** (This is not part of the standard homework. This counts for extra credit points, which are considered only after the final cutoffs have been set. Please see the course syllabus for more information.)

There is a triangle $\triangle abc$ on the $xy$-coordinate plane and there are two mosquitos, named Peter ($p$) and Rita ($r$), both of whom are on the $+z$ side of the $xy$-coordinate plane. In the manner of Problem 1, describe the geometric operations to determine whether Peter can see Rita reflected in this triangular mirror (see Fig. 1(e)).
Practice Problems for the Midterm Exam

The midterm exam will be on Thu, Oct 27 in class. The exam will be closed-books, closed-notes, but you will be allowed one sheet of notes, front and back to use for the exam. The following problems have been assembled from old exams and homeworks. They do not necessarily reflect the actual difficulty of problems on the exam or the total length of the exam. You are responsible generally for material covered in class or appearing on class assignments.

Problem 1. Short answer questions. Explanations are not required, but may be given for partial credit.

(a) In the call glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH), explain in English (in a single sentence for each) the meaning of each of the capabilities that have been enabled.

(b) In gluLookAt(), in what direction is the up vector not allowed to point. Explain.

(c) You are given a 2 × 1 rectangle with corner vertices a, b, c, and d, as shown below. Consider the points \( p = (1.5, 0.5) \) and \( p' = (1.5, 1) \).

\[
\begin{array}{c}
\text{y} \\
\text{1} \\
\text{d} \\
\text{a} \\
\text{1} \\
\text{b} \\
\text{2} \\
\text{x}
\end{array}
\text{p' \quad c}
\]

(i) Express \( p' \) as an affine combination of \( d \) and \( c \).

(ii) Is your answer to part (i) a convex combination?

(iii) Express \( p \) as an affine combination of \( a \), \( b \), \( c \), and \( d \)? (Hint: If you try to solve a linear system of equations, you are making this way too hard.)

(d) A user draws a triangle strip using GL_TRIANGLE_STRIP and gives \( n \) vertices. As a function of \( n \), how many triangles are produced? (Assuming no three collinear vertices and no duplicate vertices.)

(e) Which of the following statements are true of perspective projections? (Select all that apply.)

(i) Lines are mapped to lines

(ii) Parallelism is preserved

(iii) Midpoints are preserved

(iv) Angles are preserved (e.g., right triangles project to right triangles)

(f) Two spheres are being rendered using glutSolidSphere. One sphere is a pure diffuse reflector and the other is a pure specular reflector. Which of the two would require higher accuracy (that is, a greater number of slices and stacks) to produce a realistic rendering? Explain briefly.

(g) What is the halfway vector and why is it relevant to computing specular reflection? (Answer in a couple of sentences.)
(h) You have a triangle in 3-space, whose vertices are \( p = (p_x, p_y, p_z) \), \( q = (q_x, q_y, q_z) \), and \( r = (r_x, r_y, r_z) \). Explain how to compute a vector \( v \) that is normal to this face, and directed to the front side (the side from which the vertices appear in counterclockwise order as \( \langle p, q, r \rangle \)). You are allowed to express your answer using vector operators, such as affine combinations, dot and cross products, etc.

**Problem 2.** You are given a procedure `drawPirate()`, which draws a 2-dimensional pirate face centered at the origin and lying on the \( x, z \)-plane. (See the figure below, part (a).) The radius of the circle forming the face is 1. Your goal is to produce a sequence of drawings of the face rolling along the \( x \)-axis, but scaled down to a radius of 1/2. (See the figure below, part (b).)

![Diagram of pirate face](image)

To do this, you are to write a procedure `rollingPirate(int n, int i)` This procedure will be called \( n + 1 \) times, for \( i = 0, 1, 2, \ldots, n \). Each call draws one image. When \( i = 0 \), the pirate will be displayed upright at \( x = 10 \). As \( i \) increases, the face rotates and translates to its next position. When \( i = n \), it will undergo a full \( 360^\circ \) rotation, as shown in the figure.

Give pseudocode for the procedure `rollingPirate(n, i)`, which uses `drawPirate()` and the OpenGL matrix stack to draw the face at the desired location and rotation. On return, the Modelview matrix stack should be unchanged.

**Problem 3.** Consider a type of light called a *spot-light*. A spot-light is defined by giving a point \( p \), a vector \( \vec{v} \) (normalized to unit length), and an angle \( \theta \). The spot light illuminates any point that lies within an infinite 3-dimensional cone whose apex is \( p \) and whose angular radius about \( \vec{v} \) is \( \theta \). Write a function which, given a point \( q \) in 3-space, and \( p, \vec{v}, \) and \( \theta \), determines whether \( q \) is illuminated by the spot-light.

![Spot-light diagram](image)

**Problem 4.** In this problem you may assume that \( z = 0 \), and we are using `glOrtho2d` for viewing. Suppose that you have an OpenGL procedure `drawE()`, which draws an upper-case letter ‘E’ of height 1, so that its lower left corner coincides with the origin. Show how to achieve each of the following tasks using OpenGL. Assume that the current transformation mode is `GL_MODELVIEW`. You may call the procedure `drawE()`, but you may not modify its contents. On return, the OpenGL transformation stack should be unchanged.
(a) Give code for a procedure `drawE1(x, y, h)`, which draws the letter ‘E’ so that its lower left corner is at position \((x, y)\) (and \(z = 0\)) and it has been uniformly scaled to be of height \(h\). All three arguments are of type `GLfloat` and \(h\) is positive. Briefly explain.

(b) Give code for a procedure `drawE2(x, y, h)`, which draws an italic letter ‘\(E\)’ by slanting the letter by 30 degrees to the right. Again the lower left corner is at \((x, y)\) and the height is \(h\). (Hint: There is no OpenGL transformation which performs a shear, so you will need to derive the corresponding matrix. Recall that \(\cos 30^\circ = \sqrt{3}/2\) and \(\sin 30^\circ = 1/2\).)

Problem 5. Your boss at Fred’s Pretty-Good Graphics Corp. wants you to write a procedure to generate a rendering of a cylinder in OpenGL. The cylinder is centered along the \(z\)-axis, has a height of \(h\) units, and has a radius of \(r\) units. Because OpenGL can only display polygons, you are to split the cylinder into \(v_s\) vertical stacks (along the \(z\)-axis) and \(r_s\) radial slices (around the \(z\)-axis). (For example, in the figure we have \(v_s = 4\) and \(r_s = 8\).) Draw each face as a `GL_POLYGON`.

![Cylinder Diagram](image)

Give a procedure (in pseudocode) `void cylinder(float h, float r, int vs, int rs)`, which draws such a cylinder in OpenGL. (You may NOT use any GLUT procedures.) (For full credit, you should specify both the vertices and associated normals, so that the shading of the cylinder will be smooth. You do not need to draw the top and bottom of the cylinder.)

Problem 6. Suppose that a viewer is located at the origin \((0, 0, 0)\), and is looking along the \((-z)\)-axis. On the plane \(y = -1\), someone has put a square pizza of side length 2 centered at the point \((0, -1, -3)\). Assume that we compute a perspective projection of the pizza onto the view plane \(z = -1\).

Where is the center?

![Pizza Diagram](image)

(a) Consider a horizontal line that bisects the projected pizza. Does the projected pizza center lie on, above, or below this line?
(b) Consider a vertical line that bisects the projected pizza. Does the projected pizza center lie on, left of, or right of this line?

Give a formal justification for your answer based on your knowledge of the perspective transformation. (Hint: You do not need to know the equation of an ellipse to solve this problem. If it makes your life easier, imagine that the circular pizza is a square.)
**Midterm Exam**

This exam is closed-book and closed-notes. You may use 1 sheet of notes (front and back). Write answers in the exam booklet. If you have a question, either raise your hand or come to the front of class. Total point value is 100 points. Good luck!

**Problem 1.** (25 points; 5–10 points for each part or subpart) Short answer questions. Except where noted, explanations are not required, but may be given for partial credit.

(a) You are told that two vectors \( \vec{u} \) and \( \vec{v} \) are each of length 2. What is the relationship (if any) between the dot product \( \vec{u} \cdot \vec{v} \) and the cosine of the angle between \( \vec{u} \) and \( \vec{v} \)?

(b) True or False: OpenGL runs mostly in *retained mode* but some elements are in *immediate mode*.

(c) Which of the following may result (directly) in a *redisplay callback*? (List all that apply.)
   (i) The window has been resized
   (ii) Passive mouse motion
   (iii) glutPostRedisplay was called
   (iv) A keyboard key was pressed
   (v) A mouse button was pressed

(d) If a GL_TRIANGLE_FAN has \( n \) vertices, how many triangles does it have?

**Problem 2.** (30 points) The boys in the art department came up with a function `drawBlock()`, which draws a 3-dimensional, axis-parallel block whose base is centered at the origin and whose side lengths are 2 × 2 × 8 (see the figure below (a)).

(a) Your boss has decided that he wants to move the block so the center top is at the point \( p = (p_x, p_y, p_z) \) (see the figure (b)). Also, he wants to see the arrow on the right side, so he asks you to rotate it counterclockwise by 90° about vertical. Finally, he wants the dimensions changed to 6 × 2 × 4, so that the side with arrow remains of width 2. Explain how to use `drawBlock`, together with the OpenGL matrix operations, to achieve the desired result. (You cannot modify the function.) The matrix stack should be unchanged afterwards.

(b) Your boss wants to see what the original block looks like if it was *tipped over*, so that it falls 18° backwards towards the \(-x\)-axis. (see the figure (c)). Show how to use the `drawBlock` function to achieve this. (Be careful! The block rotates about its back edge, not about the origin.) As before, the matrix stack should be unchanged afterwards.
Problem 3. (20 points) The next version of OpenGL, called AwesomeGL, will support a new type of anisotropic surface material that changes color depending on the viewer’s position. Two RGB colors, \( C_0 \) and \( C_1 \), are given, as is a directional vector \( \vec{w} \) that runs parallel to the surface. Consider a point \( p \) on the surface that you want to color. Let \( \vec{v} \) be the normalized vector from \( p \) to the viewer, and let \( v \perp \) be the projection of \( \vec{v} \) onto the surface. If \( v \perp \) is perfectly aligned with \( \vec{w} \) (the angle between them is \( 0^\circ \) or \( 180^\circ \)) the surface color is \( C_0 \) (see the figure below (a)). If the angle is \( 90^\circ \), the surface color \( C_1 \) (see the figure below (b)). Between these two extremes, the colors change smoothly from one to the other.

(a) The viewer is at point \( q \), and the surface normal is \( \vec{n} \). Explain how to compute \( \vec{v} \) and \( v \perp \)?

(b) Explain how to modify the Phong lighting model to handle this new type of surface. (Note that only the surface color changes; after this the lighting model must be applied. You may assume a single light source.)

![Diagram](attachment:image.png)

(a) surface color is \( C_0 \) 
(b) surface color is \( C_1 \)

Problem 4. (25 points) A common way of controlling the camera in adventure games is to have the camera follow the main character, or agent. In this problem, consider a coordinate system where the \( z \)-axis points upwards, and the agent is located at a point \( p = (p_x, p_y, p_z) \).

(a) Suppose that the agent is moving in some direction given by a velocity vector \( \vec{w} = (w_x, w_y, w_z) \). We assume that motion is parallel to the ground, so \( w_z = 0 \). You are given two positive scalar parameters \( d \) and \( \theta \). You are to place the camera so it is looking at the agent, is positioned at distance \( d \) behind the agent, and is elevated upwards from the ground by angle \( 0 < \theta < 90^\circ \) (see the figure below (a)). Derive the \textit{gluLookAt} command to achieve this. Recall that the calling sequence is

\[
\text{gluLookAt}(e_x, e_y, e_z, a_x, a_y, a_z, u_x, u_y, u_z),
\]

where \( e \) is the eye location, \( a \) is the point the camera is looking at, and \( \vec{u} \) is the up vector. The up vector should be chosen so that the \( z \)-axis points up in the image.

![Diagram](attachment:image2.png)

(a)

(b) When the user pushes the joystick to the right, the agent should make a \( 90^\circ \) turn to the right. Explain how to compute the new velocity vector \( \vec{w}' \) (see the figure (b)). Its length should be the same as \( \vec{w} \).
Programming Assignment 2: Smurf vs. Zombies (Phase I)

Handed out Tue, Nov 1. The first phase must be submitted to the grader by Mon, Nov 21 (any time up to midnight). See the syllabus for the late policy.

Overview. The goal of this project is to implement a simple 3-dimensional game, called Smurf vs. Zombies, in which the player directs an adorable blue smurf amidst an evil army of deadly zombies. The final project will involve a combination the following elements:

- Processing of both keyboard and mouse inputs
- Both automatic and user-controlled camera motion
- Lighting, shading, shadows, and texture mapping
- Simple animation control (walking, jumping, falling)
- Simple physical simulation (gravity, friction, collisions)

Phase I: Implement keyboard and mouse inputs, user-controlled camera movement, rendering with light and texture mapping, motion with gravity, friction, and air resistance.

Phase II: Add multiple zombies, shadows, complex animation, collision detection, and general game control.

Phases II will be described later in greater detail. In this first phase of the project, you are to implement the following elements. As always, we allow some flexibility in how you implement your program, provided that you achieve the main learning objectives. (If you are in doubt, please check with us.)

World: (Required) The game takes place on a square platform (called the ground) that floats in space. The characters walk around on this ground platform. If they ever move off the ground, they plummet to their death. The ground is enclosed within a much larger enclosure, which is texture mapped with a skybox.

Phase I

Phase II

Figure 1: Project 2 screenshots.
Smurf: (Required) The user controls the smurf using both keyboard and mouse inputs. The smurf is moved horizontally using the arrow keys: ‘↑’ for forward, ‘↓’ for backward, ‘←’ for left, and ‘→’ for right. (If you prefer, you may instead use ‘w’, ‘s’, ‘a’, and ‘d’ for these functions.) Each press of a key induces an instantaneous impulse (i.e., a kick) to the smurf. Thus, the more frequently a key is pressed, the faster the smurf will move, up to a given maximum speed. Each impulse is generated relative to the camera position. For example, ‘↑’ generates an impulse away from the viewer and ‘←’ generates an impulse directed to the viewer’s left. Note that the impulse directions change dynamically, because the user can move the camera.

Jumping: (10%) If the user hits the right mouse button (or some other input of your choice), the smurf jumps upwards and will be pulled down eventually by gravity. Once the smurf jumps, the horizontal component of its velocity remains fixed, and the aforementioned keyboard input is ignored.

Simple physics (10%) Your program should simulate friction with the ground and gravity. Friction is applied only when the smurf is on the ground and gravity is applied only when it is in the air. (We will provide a file with further information on how we computed the effects of friction and gravity.)

Zombie: (Required) In the final project there will be many zombies, which will “spawn” randomly over time. For Phase I, you are required to implement only a single zombie. A zombie moves at a fixed horizontal speed, and turns slowly to chase the smurf. Since they turn slowly, the smurf’s principal means of defense is to lure a zombie close to the edge of the ground and quickly step out of the way, causing the zombie to fall over the edge.

Camera motion: (Required) The camera’s elevation and distance can be adjusted by the user. The camera is positioned at a fixed set of spherical coordinates relative to a point that lies on the ground immediately beneath the smurf. When the mouse is dragged up and down, the camera’s elevation increases and decreases. When the mouse is dragged left and right, the camera rotates horizontally around the smurf. Through the use of two key inputs (we used ‘o’ and ‘i’) the camera can be zoomed out or zoomed in. You should avoid allowing the elevation to get too high or too low. (We limited ours to about 85° above and below.)

For the sake of visual continuity, when the smurf jumps, the camera does not itself move upward. Instead, it continues to point to a position immediately underneath the smurf on the ground.

Ground Drawing: (10%) Since (as we will see below) the camera follows the smurf, rendering the ground as a large rectangle of uniform color (especially in the absence of shadows) makes it very difficult to ascertain the smurf’s motion. You should render the ground in some manner that makes it easy to ascertain the speed and direction of motion. In our case, we rendered it as a tiling of blocks, where the blocks are slightly shrunken to reveal small gaps between them. The gaps are just for the sake of visualization, and have no meaning in the game.

Modeling: (10%) You may model the smurf as you like, but your grade will depend in part on doing a reasonable job in making it look “smurfy.” Our smurf was rendered as a collection of spheres which were suitably scaled and translated to make a head, body, hands and feet. (You may upload a model from some other source, but remember that you must cite all external sources.) Note that in Phase II, you will be asked to add more complex animation to your model (e.g., simulating a walking motion). You may reuse the same smurf model or create a different model for your zombies, but it must be clear (e.g., by color) which is which.
**Lighting:** (Required) Your program should make use of at least one light source to illuminate elements of your scene. In Phase II you will be required to generate ground shadows for one of your light sources. Extra credit points will be given for more elaborate lighting set-ups or special effects.

**Texture mapping:** (Required) Your program must have at least one element of texture mapping, namely a texture mapped skybox. For extra credit, you may have other texture mapped entities. (We will provide a sample image and code for reading in image files and storing them in a manner that is compatible with OpenGL.) You are free to write your own code for loading images or steal something from the web. (Remember that you must cite all external sources.)

**Resources:** We will make a sample executable available on the class projects page (under “Projects”) along with some other useful files (our object file and skybox images).

**External Resources:** An important learning objective with this project (all phases) is your ability to process basic 3-dimensional geometry, rendering, animation, and simple physics (involving both linear and angular motion, collision detection, and collision response). In practice, much of this would be done with the aid of geometric modelers, a game engine, and a physics engine. However, for this project, we would like you to do as much of this as possible on your own, in order to acquire an understanding of how the internal elements of these systems work.

For this reason, other than OpenGL and GLUT, you are *not* allowed to make use of high-level software tools for tasks such as rendering, geometric modeling, or physics. However, you are allowed to download and use of small technical pieces of code (e.g., code for performing basic matrix or quaternion operations). If you are not sure whether some code satisfies our conditions of “fair use”, please check with me.
Homework 2: Perspective and Lighting

Handed out Tuesday, Oct 18. Due at the start of class Tuesday, Oct 25. Late homeworks are not accepted, so turn in whatever you have done.

Problem 1. Provide a short answer to each question. Please explain how you derived your answer.

(a) Consider the parabola \( y = x^2 \) in the projective plane. Consider the extensions of the two ends of the parabola out to infinity. What are the homogeneous coordinates of these points at infinity? (Hint: The answer may surprise you. It surprised me!)

(b) What is the principal advantage (from the perspective of efficiency) of specifying that a light source is at infinity (by setting its \( w \) coordinate equal to 0)?

(c) Recall the term for specular reflection in the Phong illumination model: \( \max(0, \vec{n} \cdot \vec{h})^\alpha \cdot L_s \), where \( \vec{n} \) is the normal vector, \( \vec{h} \) is the halfway vector, and \( L_s \) is the light color. What is the effect of increasing the \( \alpha \) term? (E.g., is the specular point larger? is it brighter? does it’s shape change?)

Problem 2. An anisotropic reflector is a surface where lighting varies directionally, that is, rotating the surface about its normal causes light to be reflected differently. (In contrast, the Phong model is isotropic, since rotating the surface about its normal has no effect on the reflected intensity of light.) A good example of an isotropic surface is brushed aluminum, as seen on modern refrigerator doors. As part of the design of the next generation of OpenGL, you have been asked to come up with a modification to the Phong lighting model to enable a simple form of an anisotropic reflecting surface.

In the standard Phong model, the parameters for the specular strength, \( \rho_s \), is fixed for the entire surface. In this case, however, the specular strength will be a function of the relationship of the light vector and a directional “grain” of the anisotropic surface.

Assume that the surface being illuminated is planar. You are given a unit length vector \( \vec{w} \) that runs parallel to the surface (see Fig. 1). Consider a point \( p \) on the surface whose shading you want to compute. Recall the light vector \( \vec{\ell} \), which runs from \( p \) to the light source. Let \( \ell^\perp \) be the component of \( \vec{\ell} \) that is parallel to the surface. (You may assume that \( \ell^\perp \) is not zero.)

![Figure 1: Problem 2.](image-url)
The way our anisotropic reflector works is this. If the angle between $\ell \perp$ and $\vec{w}$ is either 0° or 180° then the specular component is the same as would be computed by the standard Phong model. On the other hand, if the angle between the $\ell \perp$ and $\vec{w}$ is 90°, then there is no specular component to the lighting. There should be a smooth variation between these two extreme cases (which you may implement in any reasonable way you like).

Give a modified formula for the Phong lighting model to handle this type of surface. The only new parameter to the formula is the vector $\vec{w}$.

**Problem 3.** OpenGL’s local lighting model does not compute shadows. One way to produce the shadow of an object is to explicitly compute it yourself and just draw the shadows on the ground. To do this, we need to derive a function that projects each point of an object onto its shadow point on the ground. Let us consider how to do this in a simple 2-dimensional setting.

Let $L = [\ell_x, \ell_y, 1]^T$ be the homogeneous coordinates of a light source in the plane, and let $P = [p_x, p_y, 1]^T$ be a point whose shadow we want to compute (see Fig. 2). Let the $x$-axis denote the ground. Let $Q = [q_x, 0, 1]^T$ denote the shadow point cast by $P$ on the $x$-axis.

(a) Give a function that determines the coordinates of $Q$ as a function of $P$ and $L$. You may assume that $p_y < \ell_y$. (Hint: Construct similar triangles.)

(b) Express your answer to part (a) as a $3 \times 3$ matrix transformation $M$. This matrix should have the following properties. First, its entries can depend only on $L$, not on $P$ or $Q$. Second, let $Q' = M \cdot P$, and let $Q$ be the perspective normalization of $Q'$ (which comes about by dividing through by the last coordinate, that is, if $Q' = [q'_x, q'_y, q'_w]$ then $Q = [q'_x/q'_w, q'_y/q'_w, 1]$. Then, the coordinates of $Q$ should match the result you obtained for part (a).
Homework 3: Advanced Rendering

Handed out Tuesday, Nov 29. Due at the start of class Thursday, Dec 8. Late homeworks are not accepted, so turn in whatever you have done.

**Problem 1.** The purpose of this problem is to work through the details of environment mapping, but in a simple 2-dimensional context.

Suppose that you have a circle of radius 1 centered at the origin of the $x, y$-plane. An observer whose eye position is $e = (e_x, e_y)$ is observing this circle and wishes to determine the color of the surface point on the circle at coordinates $p = (p_x, p_y)$. (Thus, $p_x^2 + p_y^2 = 1$.)

You are given four 1-dimensional textures, as shown in the figure below, called $a$, $b$, $c$, and $d$. Each texture is indexed by a texture coordinate $s$, which varies from 0 to 1, where 0 is the lower/right end of the image and 1 is the upper/left end of the image.

(i) Given $e$ and $p$, explain how to compute the normalized reflection vector $\vec{r}$. (No trick. This should be easy.)

(ii) Given the coordinates $\vec{r}$, explain how to determine which of the four textures is to be accessed and which texture coordinate corresponds to this reflection vector. (Remember that in environment mapping, the vector $\vec{r}$ is always assumed to originate from the origin of the coordinate system, and not the point $p$ at which it contacts the surface.)

**Problem 2.** In Lecture 18 we presented an approach for shadow computation which involved the use of the OpenGL stencil buffer. The approach described in the lecture worked for exactly one planar surface. The purpose of this problem is to generalize this to many surfaces.

Suppose you have $n$ planar surfaces on which you wish to render shadows. (Let’s assume that $n$ is pretty large, say up to 255.) You are given a function `drawSurface(i)`, which draws the $i$th planar surface. You are also given a function `drawShadow(i)`, which draws the 2-dimensional shadow on (or actually, just slightly above) the $i$th planar surface. (Such a function computes and applies a shadow projection matrix for the $i$th planar surface, and then draws the object in the shadow color.)

Given these procedures, explain how to modify the shadow drawing procedure from Lecture 18 to produce a shadow on each of the desired surfaces. The stencil buffer should be used so that each shadow appears only on the desired surface.
You do not have to give all the code. It would be sufficient to give a high-level explanation of what changes are made, and what order are the operations performed. If you don’t give the full code, I would like you to explicitly give the exact calls affecting the OpenGL stencil buffer (glStencilFunc and glStencilOp).

**Hint:** Because it disables the depth test, the method described in class may be inaccurate if shadow surfaces overlap one another. To simplify things, you may assume that the shadows from different objects do not overlap each other in the image.

**Problem 3.** Consider a hollow cylinder, whose axis is aligned with the $z$-axis, has a radius of 1, and extends from $z = 0$ to $z = 1$.

![Diagram of a hollow cylinder](image)

(a) Give an implicit representation for the *infinite cylinder* (no constraint on $z$). That is, give a function $f$ such that $f(x, y, z) = 0$ for each point $(x, y, z)$ on the cylinder.

(b) Given a point $p$ and unit vector $\vec{u}$, consider the ray $p + t\vec{u}$. Present a pseudo-code procedure that determines the value $t$ of the first intersection of the ray with the hollow cylinder. If there is no intersection, indicate this. Note that the ray may hit the cylinder either from the inside (as in ray $(p, \vec{u})$ in the figure) or from the outside (as in ray $(p', \vec{u}')$).

Explain how you derived your answer. (Hint: It may be useful to recall the quadratic formula, which states that the roots of $ax^2 + bx + c = 0$ are $(-b \pm \sqrt{b^2 - 4ac})/2a$.)

(c) Give a procedure which determines the normal vector at the point of intersection. The normal should be of unit length and directed to the same side of the cylinder from which the ray hits.
Programming Assignment 2: Smurf vs. Zombies (Phase II)

Handed out Tue, Nov 22. Due Mon, Dec 12 (any time up to midnight). See the syllabus for the late policy.

Overview. This is the second phase of the Smurf vs. Zombies project. Recall the overall goals of the project:

- Processing of both keyboard and mouse inputs
- Both automatic and user-controlled camera motion
- Lighting, shading, shadows, and texture mapping
- Simple animation control (walking, jumping, falling)

In this phase of the project you will add the following elements to Phase I: multiple zombies, shadows, complex animation, collision detection, and general game control. Here is a summary of the requirements.

Phase I requirements: Because many key functional aspects of Phase I are needed in this phase, we will expect you to fix any unimplemented elements of Phase I that are required in Phase II. This includes mouse/keyboard based camera control, camera-based smurf control and jumping, and zombie slow-turning behavior. (Your implementation need not match ours exactly, provided that the overall game still has the general “look and feel” of our implementation.)

Multiple Zombies: (Required) Your program should “spawn” some number of zombies. They should appear at random locations along the ground, and with initial random directions. You may have them spawn randomly over time (e.g., on average one per second) or at regular time intervals (e.g., exactly one zombie each second). But, they should not all appear at once.

We put an absolute limit on the number of zombies, so it is possible for the game to end. You are not required to do this, but there should be some identifiable end-of-game state. (For example, maybe your smurf needs to visit a certain set of special squares before getting eaten by a zombie.)

Shadows: (20%) Your smurf and zombies should cast shadows on the ground. Objects do not need to cast shadows onto each other nor onto the sides of ground blocks, only on to the ground’s upper surface. Shadows should be cast accurately with respect to a nonvertical light source position (which may be a point at infinity). For full credit, shadows should not hang over the ends of the ground blocks. For partial credit (10% deduction) they are allowed to overhang. (Use the stencil technique described in Lecture 18.)

Complex animation: (Required)

Walking: To create a better illusion of walking, have your smurf and zombies adopt an animation style that suggests walking behavior. For example, we implemented having the feet rotate rhythmically about an invisible hip joint and had the smurf wobble from side to side in cadence with the feet. You are free to come up with an alternate motion, but for full credit, it should be at least as convincing. Ideally, your walking motion should be linked to speed, so that faster motion results in faster foot movement.
**Facing:** Your smurf and zombies should face the direction they are walking. This should be done gradually over time. For example, we maintained two angles, one is the angle the entity is facing and the other is the direction it is moving. If the two angles are not the same, the facing angle is slowly turned in the direction of the motion angle.

**Spawning Zombies:** Rather than just have the zombies appear suddenly, find a way to have them appear slowly over time. (To give your smurf time to get out of the way.) We rotated them up from a horizontal position, but you can adopt another approach. While in this special spawning state, the zombie does not move toward the smurf and it cannot yet attack the smurf.

**Falling:** When a smurf or zombie falls over the edge, create some different animation. We had them rotate end over end.

**Collision detection:** (Required) Detect whether your smurf collides with a zombie. (This can be as simple as checking that their \((x, y)\) coordinates are sufficiently close.) If so, the smurf either dies or suffers damage (in our case, the smurf dies and the game ends.) In our implementation a smurf cannot collide with when it is jumping.

**General game control:** (20%) You should detect the end state of the game. (For example, if all the zombies fall off the ground, the smurf wins. If the smurf collides with a zombie or falls off the ground, the zombies win.) Rather than just closing the program, it should enter some state that will allow the user to gracefully quit the program or restart the game.

In our game, we changed the background color, freeze the game, and printed a message. You are allowed to chose some other method. It should be clear whether the game was won or lost by the smurf.

For extra credit, consider implementing a paused state, where the animation freezes, but the camera control still operates.

As always, you may (and are encouraged to) modify the technical details described above, provided that you do so in a manner that retains the same general level of complexity and maintains the general spirit of the game. If you are unsure, feel free to check with me.
Practice Problems for the Final Exam

The final will be on Mon, Dec 19, 10:30am–12:30pm. The exam will be closed-books, closed-notes, but you will be allowed two sheets of notes, front and back to use for the exam. The following problems have been assembled from old exams and homeworks. They do not necessarily reflect the actual difficulty of problems on the exam or the total length of the exam. Also, do not forget to review material from before the midterm.

Problem 1. Short answer questions.

(a) For each of the following operations, indicate which OpenGL buffer is most relevant to the operation (just list one): Color buffer, depth buffer, accumulation buffer, or stencil buffer.
   (i) Blending and motion blur
   (ii) Hidden surface removal
   (iii) Lighting and shading
   (iv) Masking

(b) The cross product of two unit vectors \( \vec{u} \times \vec{v} \) is of unit length. What can be said about their dot product, \( \vec{u} \cdot \vec{v} \)? (Pick one.)
   (i) It must be 0.
   (ii) It need not be 0, but it must be a number between \(-1\) and \(+1\).
   (iii) It will be either \(-1\) or \(+1\), but we cannot determine which.
   (iv) Nothing. The cross and dot product are unrelated to each other.

(c) What is Lambert’s Cosine Law? Explain briefly how this law is used to compute the diffuse illumination term in the Phong model:

\[
\max(0, (\vec{n} \cdot \vec{\ell})) L C_d.
\]

Recall that \(\vec{n}\) is the surface normal, \(\vec{\ell}\) is the unit vector to the light source, \(L\) is the color of the light, and \(C_d\) is the diffuse color of the object.

(d) Explain the difference in how smooth shading is performed in Phong shading and Gouraud shading. Which method does OpenGL use?

(e) What is the inverse texture wrapping function, and why is it more relevant to the rendering process than the texture wrapping function?

(f) A ray is shot at a transmissive and nonreflective surface, and total internal reflection occurs. From which side did the ray strike? (i) the one of higher IOR (index of refraction), (ii) the one of lower IOR, or (iii) could be either.

(g) In ray tracing, whenever a ray arrived at a surface, we shot rays to each of the light sources. What was the purpose for doing this?

(h) Give the three blending functions for a Bézier curve of order 2.

(i) B-splines possess a property called local support. What is local support, and why is this property desirable?

Problem 2. An important utility is drawing text strings. This question will consider doing this, assuming you have access to a function that draws individual characters. Assume that each character is defined by an integer code (e.g., its ASCII or Unicode value). You are given the following:
• The function \( \text{draw}(i) \) draws the character whose character code is \( i \), so that its lower left corner is at the origin. (See the figure below (a).)
• Different characters have different widths. You are given an array of widths, where \( \text{width}[i] \) holds the width of the \( i \)th character.
• You are given a character string to draw, where \( \text{string}[j] \) contains the character code of the \( j \)-th character to be drawn, and \( \text{string}.\text{length} \) is the number of characters in the string.

\[
\text{drawString1}(\text{"LEFT"})
\]

\[
\text{drawString2}(x_0, y_0, 1.5, 20, \text{"LEFT"})
\]

(a) Use the above and OpenGL transformations (\text{glTranslatef()}, \text{glRotatef()}, etc.) to implement a procedure \( \text{drawString1(string)} \) that draws the entire string on the \((x, y)\) plane so that its lower left corner coincides with the origin (see figure (b)). You may not use any of the built-in OpenGL or GLUT commands for drawing strings.

(b) Implement a procedure \( \text{drawString2}(x_0, y_0, s, \theta, \text{string}) \) that draws the given string so that its lower left corner is at the point \((x_0, y_0)\), it is scaled uniformly by the factor \( s \), and rotated by angle \( \theta \), given in degrees (see figure (c)). Except for the string, all the arguments are of type \text{GLfloat}. (Hint: You may call your procedure from part (a).)

**Problem 3.** In this problem we derive the implicit and parametric representations of a cylinder. Consider an infinite cylinder of radius \( 1/2 \) centered whose central axis is parallel to the \( x \)-axis, and which passes through the point \((0, 2, 1)\).

(a) Give an implicit function representation of this cylinder, by giving a function \( f \) such that \( f(x, y, z) = 0 \) for each point on the surface of the cylinder.

(b) Present a parametric representation for the same cylinder, e.g. as \( x(u, v), y(u, v), z(u, v) \). What are the range of values for \( u \) and \( v \)?
Problem 4. Consider the cone shown in the figure below. Its axis is the $z$-axis, its apex is at the origin, and its base has radius $r$ and is located at $z = 3$. We wish to wrap a rectangular texture around the central third of the cone. (Thus the bottom edge of the texture coincides with $z = 1$ and the top edge coincides with $z = 2$.) As $s$ varies from 0 to 1, the texture should make one full revolution around the cone, starting from directly above the $x$-axis.

(a) Give a parametric representation of the cone. That is, given two real parameters $u$ and $v$, show that any point $p = (x, y, z)$ that lies on the cone can be expressed as $x(u, v)$, $y(u, v)$ and $z(u, v)$, for some choice of $u$ and $v$.

(b) Given your parameterization from (a), what are the ranges of values for $u$ and $v$ in order to generate the above cone?

(c) Give the inverse wrapping function, which maps a point $(x, y, z)$ on the central third of the cone the corresponding point $(s, t)$ in texture space.

Problem 5. Fog is a relatively easy enhancement to a ray tracer. Fog is defined by three parameters, fogStart, fogEnd, and the fog RGB color $F$. Let $C$ be the color returned by the ray tracing procedure (ignoring fog). Let $d$ be the distance from the ray origin to the point of contact. If $d$ is less than fogStart then $C$ is used, if $d$ is greater than fogEnd then $F$ is returned. Otherwise, an appropriate mixture of the two colors is returned. Give pseudocode for a function, which returns the fog color, given the following parameters: the ray origin $p$, the ray contact point $q$, the traced color $C$, and the other fog parameters fogStart, fogEnd, and $F$.

Problem 6. Write a procedure to test whether a ray $p + t\vec{u}$, for $t > 0$, intersects a rectangle lying on the $z = 0$ plane, whose corner coordinates are $(-1, -1, 0)$ and $(+1, +1, 0)$. If the ray does not intersect, then the procedure should return special value MISS to indicate this, and otherwise it should return the $t$-value of the intersection point.

Problem 7. You have been asked you to produce a ray-intersection procedure for cereal-bowl shape. The cereal bowl is the bottom-half of a unit sphere, which is centered at the origin. Assume that the $z$-axis points up.
(a) Let \( p \) be a point and \( \vec{u} \) be a unit vector. Given a ray \( p + t\vec{u} \), present a procedure (as either mathematical formulas or pseudo-code) that determines the value \( t \) of the first intersection of the ray with the bowl. If there is no intersection with the bowl, your program should detect this case. Explain how you derived your answer. (Hint: It may be useful to recall the quadratic formula, which states that the roots of \( ax^2 + bx + c = 0 \) are \( (-b \pm \sqrt{b^2 - 4ac})/2a \).)

(b) Give a procedure which determines the normal vector at the point of intersection. The normal should be of unit length and directed to the same side of the bowl from which the ray hits.

Problem 8. In this problem, we will consider how to render a scene with a mirror in OpenGL. You have a procedure, \textit{drawScene()}, which draws a given scene. You may assume that the scene resides entirely in the positive \((x, y, z)\)-orthant (that is, \( x \geq 0 \), \( y \geq 0 \), and \( z \geq 0 \) for all objects in the scene). Imagine that on the \( y = 0 \) plane, there is a rectangular mirror, as shown in the figure below, which ranges from \([1, 2]\) on the \( x \)-axis and \([1, 4]\) on the \( z \)-axis. You are to write an OpenGL procedure to render this scene and its reflection in the mirror through the use of the stencil buffer. The mirror is a perfect reflector, and hence no color blending is required.

(a) Give a step-by-step high-level description of how this will be done. You do not need to give specific OpenGL commands, but it should be clear how to translate your ideas into OpenGL operations (e.g., “save the matrix state”, “disable the depth test”, “draw a polygon with vertices . . . ”, etc.). It should be clear from your answer what order the operations are to be performed, what specific transformations are to be applied, what shapes are to be drawn, etc. In this part you may ignore lighting.

(b) If lighting is to be applied to the objects rendered in the mirror, should the light positions be modified, and if so, how?

Problem 9. Let \( b_{k,d}(u) \) denote the \( k \)-th Bézier blending function of degree \( d \). Recall that given an array \( p_{ij} \), of control points \( 0 \leq i, j \leq 3 \), the Bézier surface is given by

\[
p(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} b_{i,3}(v)b_{j,3}(u)p_{ij},
\]

for \( 0 \leq u, v \leq 1 \). Given any fixed \( u_0, 0 \leq u_0 \leq 1 \), define the \( u_0 \)-slice to be the curve \( s(v) = p(u_0, v) \). Show that \( s(v) \) is a Bézier curve of degree 3. What are the four control points for this curve?
Final Exam

This exam is closed-book and closed-notes. You may use 2 sheets of notes (front and back). Write all answers in the exam booklet. If you have a question, either raise your hand or come to the front of class. Total point value is 100 points. Good luck!

Problem 1. (30 points; 2–8 points each) Short answer questions. (Explanations are not required, but may be given for partial credit.)

(a) Given two vectors $\vec{u}$ and $\vec{v}$ in 3-dimensional space, you are told that $\vec{u} \times \vec{v} = 0$. Which of the following can you infer from this? (List all that apply.)

(i) One or both vectors must be the zero vector, $(0, 0, 0)$.
(ii) If the vectors are both nonzero, they are perpendicular to each other.
(iii) If the vectors are both nonzero, they are parallel to each other.
(iv) The dot product $(\vec{u} \cdot \vec{v})$ must also be zero.

(b) Answer the following questions involving the function `gluPerspective(fovy, aspect, near, far)`.

(i) If the ratio $\text{far}/\text{near}$ is made unnecessarily large, what sort of error could arise in the final image?

(ii) Suppose you wanted to produce an effect of zooming in to produce a close-up of an object at the center of the image. Which parameter would you change? Would you decrease or increase its value?

(iii) Suppose that you have a viewport whose lower left corner is at $(5, 10)$ and whose width and height are 20 and 50, respectively. What would you set the aspect parameter to be (assuming you want no distortion)?

(c) (True or false?) In 3-dimensional affine geometry, the homogeneous coordinates $(3, 4, 5, 0)$ and $(-3, -4, -5, 0)$ represent the same object.

(d) (True or false?) In 3-dimensional projective geometry, the homogeneous coordinates $(3, 4, 5, 0)$ and $(-3, -4, -5, 0)$ represent the same object.

(e) (True or false?) In OpenGL, if a white object is placed very close to an object with a high red emission value (set using `glMaterial(GL_FRONT_AND_BACK, GL_EMISSION, ...)`), parts of the white object may take on a reddish shade.

(f) Two spheres are being rendered using `glutSolidSphere`. One sphere is a pure diffuse reflector and the other is a pure specular reflector. Which of the two would require higher accuracy (that is, a greater number of slices and stacks) to produce a realistic rendering? Explain briefly.

(g) You are given a parametric representation of a surface $p(u, v) = (x(u, v), y(u, v), z(u, v))$, and you can compute the derivatives of these functions. Explain how to compute a surface normal vector at a particular point $p(u_0, v_0)$. 

1
Problem 2. (10 points) Given three control points \( p_0, p_1, \) and \( p_2 \) in the plane, recall that the Bézier curve of degree-2 is:
\[
B(u) = (1 - u)^2 p_0 + 2u(1 - u)p_1 + u^2 p_2.
\]
Prove the following claim: For any three points \( p_0, p_1, \) and \( p_2 \), the tangent to the curve at \( u = 1/2 \) is parallel to the line segment \( \overline{p_0p_2} \) (see the figure below). Prove this from first principles. You may not use any facts about Bézier curves given in class.

(Hint: Start by computing the derivative of \( B(u) \), carefully.)

Problem 3. (20 points) You are given a cylinder of radius 1 aligned with the \( z \)-axis, running from \( z = 0 \) to \( z = 2 \). You are given a texture map whose \( s \) and \( t \) coordinates range from 0 to 1. You are to wrap this texture around the cylinder so that the central third of the texture \((1/3 \leq t \leq 2/3)\) appears like a “candy-cane stripe”. The stripe should start above the \( x \)-axis, and wrap exactly once around the cylinder (see the figure below).

In this problem you will derive the inverse wrapping function, which maps a point \((x, y, z)\) on the cylinder to the corresponding point \((s, t)\) on the texture.

(a) Give a parametric representation of the cylinder. That is, given two real parameters \( u \) and \( v \), show that any point \( p = (x, y, z) \) that lies on the cylinder can be expressed as \( x(u, v), y(u, v) \) and \( z(u, v) \), for some choice of \( u \) and \( v \).

(b) Given your parameterization from (a), what are the ranges of values for \( u \) and \( v \) in order to generate the cylinder?

(c) Give the inverse wrapping function, which maps a point \((x, y, z)\) on the cylinder to the corresponding point \((s, t)\) in texture space. (Don’t worry about the gray areas. I only care that your function works for the points on the stripe.)

Problem 4. (20 points) You have been asked you to produce a ray-intersection procedure for an object called a capped cylinder, which consists of a cylinder that extends infinitely far downwards and has a hemispherical cap on its top (see the figure below). Assume that the cylinder’s axis coincides with the \( z \)-axis, its radius is 1, and it extends infinitely down along the negative \( z \)-axis. The cap on top is an upper hemisphere of radius 1 centered at the origin.
Let $p$ be a point and $\vec{u}$ be a unit vector. Given a ray $p + t\vec{u}$, present a procedure (as some combination of mathematical formulas and pseudo-code) that either determines the point of first intersection of the ray with the capped cylinder or else indicates that no such intersection exists. Note that the origin of the ray may be inside the capped cylinder. Explain how you derived your answer. (Hint: It may be useful to recall the quadratic formula, which states that the roots of $ax^2 + bx + c = 0$ are $(-b \pm \sqrt{b^2 - 4ac})/2a$.)

Problem 5. (20 points) In this problem, we will consider how to render objects with a reflective surface in OpenGL. First, you have a table, which is a horizontal (filled) rectangle that floats above the ground at height $z = 10$. The function drawTable() draws the table. On the table is a puddle of spilled water, which behaves like a perfect reflector. It is drawn using the function drawPuddle() (see part (a) of the figure below). Finally, there are some objects that sit on the table (a sphere and a pyramid shown in part (b) of the figure below), which are drawn by a function drawObjects().

(a) Give a function drawReflectedObjects(), which invokes drawObjects() so that the objects are drawn as if they have been reflected below the table top (see part (c) of the figure). The matrix state should be unchanged on returning from this function.

(b) Give a function to render the entire scene: table top, objects, and puddle with reflected objects. You are not required to give specific OpenGL commands, but it should be clear how to translate each of your operations directly into OpenGL (e.g., “save the matrix state”, “disable the depth test”, “draw a given shape into the stencil buffer with reference value . . .”, etc.). It should be clear from your answer what order the operations are to be performed, what specific transformations are to be applied, what shapes are to be drawn, etc.

You may ignore lighting issues throughout.