Homework 1: OpenGL and Data Structures

Handed out Saturday, Feb 23. Due at the start of class Thursday, Feb 28. Late homeworks are not accepted, so turn in whatever you have done.

Problem 1. Short answer questions. Except where noted, explanations are not required, but may be given for partial credit.

(a) You start with the Modelview matrix stack in its default state (just the identity matrix) and then you execute (in sequence) glLoadIdentity(), gluLookAt(), glPushMatrix(), and glTranslatef(). How many matrices are now on the Modelview stack? How many matrix multiplications have been performed?

(b) Which of the following events might cause the system to signal a *redisplay event*? (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

(c) If n vertices are given as part of each of the following OpenGL objects, how many triangles result?

(i) GL_TRIANGLES
(ii) GL_TRIANGLE_FAN
(iii) GL_TRIANGLE_STRIP

(d) It is fact that the Delaunay triangulation of a set of points in the plane is a 2.418-spanner. (To be precise, it is a $(4\sqrt{3}/9)\pi$-spanner). What does this mean?

Problem 2. You have a main graphics window of width w and height h, such that $h < w < 2h$ (see Fig. 1). We want to draw two square viewports that we want to serve as viewports. The first viewport is the same height as the main window and is aligned with the main window's right edge. The second viewport is aligned with the upper left edge of the main window, and is as large as possible, without overlapping the first viewport.

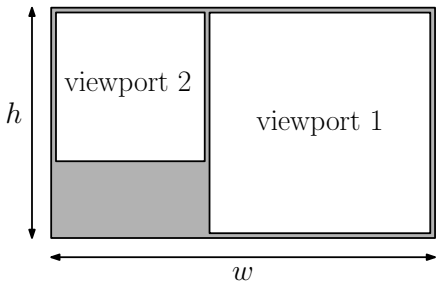


Figure 1: Problem 2.

Given the commands to glViewport for these two viewports. Recall that the calling sequence is:

glViewport(x, y, vw, vh);

where (x, y) are the coordinates of the lower left corner of the viewport (where the origin is in the lower left corner of the main window, relative to the lower left corner of the main window), and vw and vh are the width and height, respectively, of the viewport.

Problem 3. 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 in 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 its height is h. All three arguments are of type GLfloat and h is positive. Briefly explain.

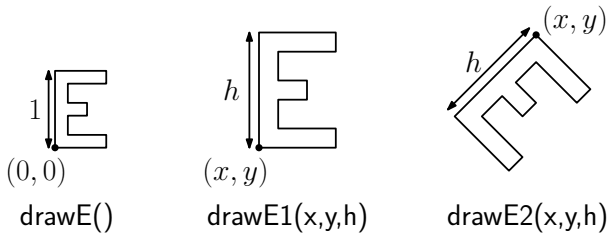


Figure 2: Problem 3.

(b) Give code for a procedure drawE2(x, y, h), so that its upper left corner is positioned at (x, y) and it has been rotated 45° clockwise.

Problem 4. You are given a 2-dimensional cell complex that is represented as a doubly-connected edge list (DCEL). Given a vertex v in this cell complex, the *link vertices* are the vertices (other than v) that lie in the faces incident to v (see Fig. 3(a)). Another way to view these vertices is to imagine that we remove v from the cell complex and all its incident edges. This results in a face that contains v in its interior (see Fig. 3(b)).

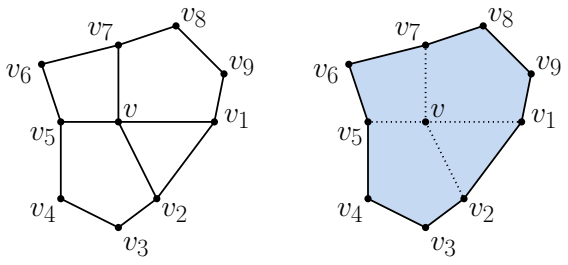


Figure 3: Problem 4.

Given the vertex v, give the code for a procedure linkVerticesCW(v) that outputs the vertices of v's link in *clockwise order* (starting at whatever vertex you like). For example, in the above example, the output might be $\langle v_1, \ldots, v_9 \rangle$. Your procedure should not modify the DCEL.

Programming Assignment 1: Space Invaders

Handed out: Tue, Feb 5. **Due:** Wed, Feb 20, 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. Submit projects by uploading to the submit server.

Overview: The goal of this assignment is to learn the basics of OpenGL and GLUT by implementing a simple two-dimensional game. The game is a highly simplified version of the ancient and venerable *Space Invaders* (and popular spin-offs like *Galaxian*). You have considerable flexibility in what game you finally implement (e.g., changing the user interface or modifying the game's behavior), as long as your game involves implementing at least as many of the basic elements as the one described here.

The game takes place inside a 2-dimensional window. Along the bottom edge of the window is a laser cannon. The cannon can be positioned horizontally to the left or right by keyboard commands (e.g., 'a' for left and 'd' for right). At the top of the window are a number of rows of aliens, which slide back and forth to the left and right. At random intervals, the aliens shoot some sort of nasty green slime down at you, which will kill you if it hits you. By clicking the left mouse button, you can shoot laser bullets up at the evil aliens. Your objective is to zap all the aliens. (The original game had quite a few more elements: the aliens moved slowly downwards, there were barricades that you could hide behind, the aliens moved progressively faster. You are welcome to add these elements for special credit.) You can use primitive shapes to represent the various objects (triangles, rectangles, etc.)

We will post a demo program later, but you can find videos of the Space Invaders arcade game to get a feeling for the game's general look. Here are some features that you should implement:

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. On termination, the game should not simply close the window. There should be a final screen, indicating whether the player has won or lost. (For extra credit, you could compute some sort of score and print that.)

Text: Your program should produce some text output to the screen. This can take various forms, e.g., printing a score at the top of the page, or messages when the game starts or ends.

Playability: Assuming a reasonable window size, it should not be too difficult for someone (even an incompetent game player like me) to play your game long enough to be able to test out your game's various capabilities. (For example, if you implement something really cool that only shows up on level 31 of your game, please give us a shortcut for accessing that level.)

Regulated animation speed: I would recommend using glutIdleFunc() to continuously update the state of your game. (You can also use glutTimerFunc().) The speed of the resulting game may depend on your monitor's refresh rate. In order to maintain a consistent speed, it is a good idea to use a "wall-clock" function like the system function ftime() to track the elapsed time between frames. See the class web page for further information.

Final Submission: Submissions will be made through the 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 2012"), 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 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 program).
- **Omit:** Microsoft Visual Studio generates ridiculously large auxilliary files in the compilation process, which we don't need. Please omit (especially large) binary files that are generated by the compiler. 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 Language: I would encourage you to use C or C++. There is a version of OpenGL for Java (jogl). You are welcome to use this, but it is up to you to figure out how to install and use it. If you want to use any other programming languages, you must clear it with the TA at least a week in advance of the due date.

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: Your grade will be determined based on whether you complete all the required elements. However, we will give extra credit points if you implement additional features, such as better graphics or more interesting game play. (See the course syllabus on how extra credit points are counted.)

Tips:

Units: Decide which units you want to use in representing your world. Remember that OpenGL may not honor your request for the window size, so your program should be able to handle (in a reasonable manner) windows of various sizes.

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., idle event), update the state. Based on the amount of time Δ_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.

External Resources: If you make use of any external resources in your program, even if you modified it, you must indicate this in your ReadMe file. (Doing otherwise would be considered plagiarism.) You are required to implement most of the program elements on your own. However, I do not mind if you make use of small pieces of ancillary code in order to implement additional features that enhance your game. If you are unsure, check with me.

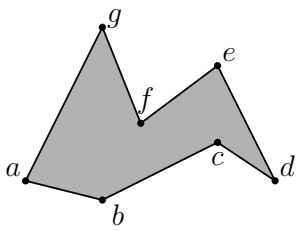
First Midterm "Quiz"

This "quiz" 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 50 points. Good luck!

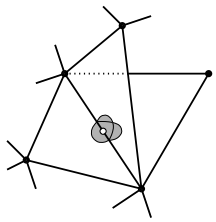
Problem 1. (15 points; 2–5 points each) Short answer questions. Except where noted, explanations are not required, but may be given for partial credit.

(a) What is *double buffering*? What is its principal advantage over single buffering?

(b) Consider the polygon shown in the figure below. Give the OpenGL calls in order to draw this as a GL_TRIANGLE_STRIP. For half credit, any triangle strip suffices. For full credit, the first triangle of the strip should be drawn in counter-clockwise order, and no vertices should be repeated. (Don't worry about the exact OpenGL syntax. Mainly I want to see that you have the right general structure and you draw the vertices in the proper order. If you do not think that it is possible to draw it as a single triangle strip, then explain why.)



Problem 1(b)



Problem 1(e)

(c) Let M be the matrix on top of the current matrix stack. Explain what the command *glRotate* does. In particular, this command constructs a rotation matrix R and modifies the matrix stack in some way. Explain exactly what these modifications are. (E.g., how does the top of stack change? is something new pushed on the stack? is the stack popped? if matrices are multiplied, what is the order of multiplication?)

(d) The command gluPerspective allows the user to specify the aspect ratio. Suppose that your graphics window has an aspect ratio of α_1 and you are drawing to a viewport with an aspect ratio of α_2. What should you set the aspect ratio argument to in gluPerspective so that the image is not distorted?

(e) In a triangulated mesh, three triangles come together at a common edge (see the above figure). Based on this information alone, what can be said about the mesh? (List all the apply.)

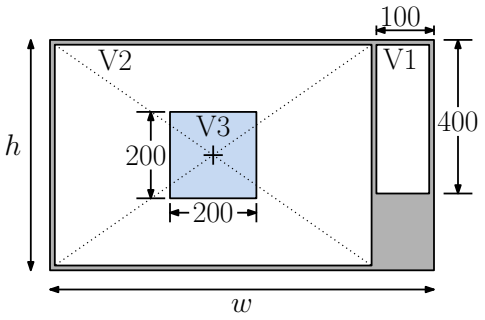
(i) it is *not* a cell complex

(ii) it is *not* a simplicial complex

(iii) it is *not* a 2-manifold

(iv) a DCEL could *not* represent it

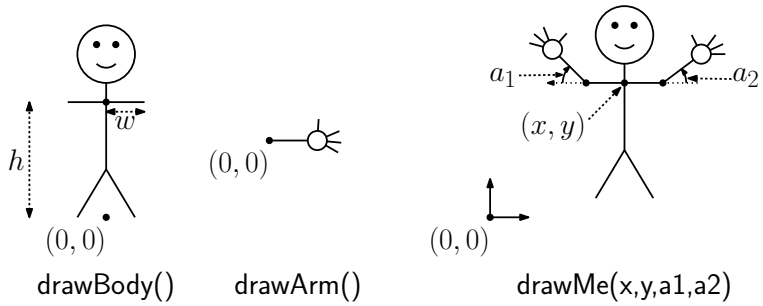
Problem 2. (15 points) As a new employee at Awesome Games Inc. your first job is to design the viewport structure for their latest flight simulator game. You are given a graphics window of width w and height h, where both w and h are at least 500 pixels. The screen will consist of three viewports. The first, viewport, V1, is a control panel of width 100 and height 400 that that is placed in the upper right corner of the graphics window. The second viewport $V2$ is positioned at the upper left edge of the graphics window and is as large as possible without overlapping V1. Finally, viewport V3, which is used for a heads-up display, is of size 200×200 and is placed in the center of viewport V2. (See the figure below.)



Problem 2.

Give the commands to glViewport for these three viewports. Recall that the form is glViewport(x, y, vw, vh), where (x, y) are the coordinates of the lower left corner of the viewport (where the origin is in the lower left corner of the main window, relative to the lower left corner of the main window), and vw and vh are the width and height, respectively, of the viewport.

Problem 3. (10 points) Suppose that you have two OpenGL drawing functions. The first one, drawBody(), draws a character's body in 2-dimensional space, where the origin lies between the character's feet immediately below its body (see the figure below). Let h be the distance from the origin to the body's shoulders, and let w denote the distance from the center of the shoulder to the elbow joint. The second function, drawArm(), draws the character's forearm so that the runs along the x-axis and its endpoint (that is, the elbow) is located at the origin.

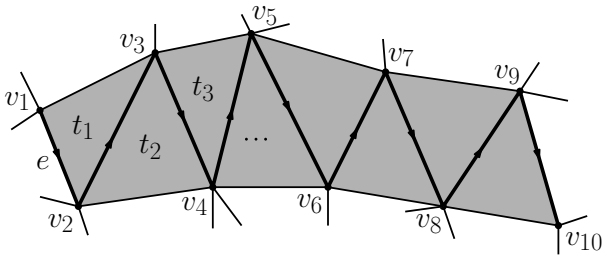


Problem 3.

Give the OpenGL code for a function drawMe(x, y, a1, a2), which does the following. It draws the body so that the center of the shoulder is situated at (x, y), and the forearms are drawn so the left one (from the viewer's perspective) is rotated a_1 degrees clockwise and the right one is a_2 degrees counterclockwise. (Beware: The hand is not symmetric. If both angles are 0, both thumbs should be pointing upwards.) You may not modify the internals of the two drawing functions, and the state of the matrix stack should be unchanged on return from your function.

Problem 4. (10 points) An important utility in efficiently rendering triangulated meshes is breaking them up into triangle strips. In this problem, we'll consider the simpler question of how to compute a single triangle strip, starting from an edge.

You are given a 2-dimensional simplicial cell complex (i.e., a triangulated mesh) that is represented as a doubly-connected edge list (DCEL). A directed edge e defines a unique triangle strip as follows. Let v_1 and v_2 be e's origin and destination vertices, respectively. Let t_1 be the triangle to e's left, and let v_3 be the other vertex of this triangle. After v_1 and v_2, follow a zig-zag pattern of diagonals, as shown in the figure below. Each vertex visited along this zig-zag is another vertex on the triangle strip.



Problem 4.

Give the code for a function getStrip(e), which given a directed edge e of the DCEL, outputs the first 10 vertices of the resulting triangle strip. (You may assume that the mesh is triangulated, and there exists 10 vertices in the strip.) For example, for the above figure, the output would be $\langle v_1, v_2, \ldots, v_{10} \rangle$. Your procedure should not modify the DCEL.

Homework 2: Graphics, Animation, and AI

Handed out Sunday, Apr 21. Due at the start of class Tuesday, Apr 30. Late homeworks are not accepted, so turn in whatever you have done.

Problem 1. Short answer questions. Except where noted, explanations are not required, but may be given for partial credit.

(a) When doing texture mapping, suppose that you want the textured surface to be subject to lighting due to the light sources. What OpenGL option would you set to cause this?

(b) Same as (a), but suppose that you *do not* want the object to be subject to lighting.

(c) Suppose that you and your friend are moving a refrigerator up a flight of stairs. You find that the refrigerator is a bit too deep, but if you swing the door open, it may be possible to angle the refrigerator through the stairs. At different points in the process, the door may need to be swung more open or less opened. (There is only one door.) How many configuration-space dimensions would you need in order to accurately model the position of the refrigerator at any time? Briefly explain.

(d) Consider an instance of A* search, and let $h(u)$ be any admissible heuristic for this instance.

(i) Is $h'(u) = h(u) + 100$ admissible? (definitely yes, definitely no, it depends on the graph?) Briefly explain your answer (using the definition of admissibility).

(ii) Suppose that you used $h'(u)$ in A* search, rather than $h(x)$. Will the result that is returned guaranteed to be correct? Briefly explain.

(Beware: The answer is subtle.)

Problem 2. In this problem we consider the performance of Dijkstra's algorithm and A* search on the graph shown below (see Fig. 1), where the path starts at s and ends at t. Each edge (u, v) is undirected and is labeled with its associated weight $w(u, v)$.

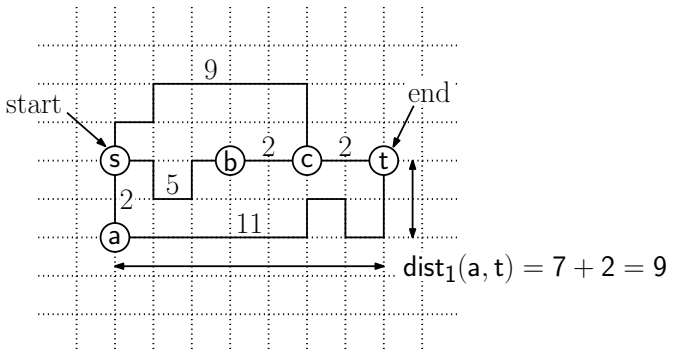


Figure 1: Problem 3.

For A* search we need to make use of a heuristic. To save you from dealing with square roots, we will use a different notion of geometric distance. Define the L_1 (or Manhattan)

distance between two points to be the sum of the absolute values of the difference of the x and y coordinates. For example, in the figure the L_1 distance between nodes a and t is $\text{dist}_1(a,t) = 9+2 = 11$. For A* search define the heuristic value for each node u to be L_1 distance from u to t. For example, $h(a) = 11$. (For this graph it is easy to verify that $h(\cdot)$ is an admissible heuristic.)

(a) Trace the execution of Dijkstra's algorithm on this graph. For each node indicate the following: (1) list the nodes in the order in which they are processed, (2) whenever a node is processed, indicate which of its neighbors have had their d-values modified, when the algorithm terminates (that is, when t is considered for processing), indicate the final d-values are for all the nodes. If there are ties for which node is to be processed next, select the node that is earliest in alphabetical order. (As an example, see Lecture 18.)

(b) Trace the execution of A* Search on this graph. Provide the same information as in (a), but also provide the A* f-values for each node that is processed (recall that $f(u) = d[u] + h(u)$).

Problem 3. In class, we showed how to compute the velocity obstacle for a pair of moving disks in the plane. In this problem, we will consider how to construct them for different shapes. Consider the two rectangles a and b, shown in the figure below (see Fig. 2). (More precisely, a's lower left corner is $(2,-2)$ and its upper right corner is $(4,0)$. Also, b's lower left corner is $(5,2)$ and its upper right corner is $(9,4)$.)

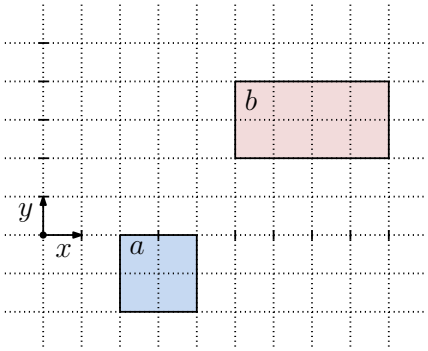


Figure 2: Problem 3.

(a) Define $\text{VO}_{a|b}$ to be the set of all velocity vectors v such that, if rectangle a moves with this velocity, it will intersect b at some time in the future. Assume that b is stationary. Describe this set of vectors as a geometric shape. (If you like, you can draw a figure to illustrate it, but be sure that your figure indicates clearly the placement of vertices or the slopes of any unbounded sides.)

(b) Suppose that we are only interested in the time interval $\tau = [0,2]$. Define $\text{VO}^{\tau}_{a|b}$ to be the set of velocity vectors such that, if rectangle a moves with this velocity, it will intersect b within the time interval τ. Describe this shape.

(c) Suppose that b is moving with velocity $v_b = (2,-1)$. That is, at time t it is translated by a distance $(2t,-t)$. How does this modify the velocity obstacle $\text{VO}_{a|b}$?

Programming Assignment 2: The Adventures of Tommy the Terp (Preliminary)

Handed out Thu, Mar 14. Due at 11:59pm on Wed, Apr 3. See the syllabus for the late policy.

Overview. The goal of this project is to implement some of the basic elements of a simple 3-dimensional interactive game (but there is not really any interesting game play). In particular, the elements that you will learn will involve

- Processing of both keyboard and mouse inputs
- Both automatic and user-controlled camera motion
- Advanced rendering using lighting and texture mapping
- Simple animation

Here is a summary of the major requirements of the project.

Tommy: The project's main moving character is named *Tommy the Terp*. He walks around within an environment of your design. At a minimum, Tommy should be able to walk along the ground, directed by inputs from the keyboard (e.g., "w a s d"). You may add additional motion elements for extra credit (jumping, for example). Tommy's model should be done in such a way that he has a clear front side, and his front side should always be rotated to face the direction he is moving.

World: The world in which Tommy walks around is entirely of your own design. At a minimum, there should be a ground surface on which he walks, but the other elements of the world are up to you.

Camera motion: One feature of the project is that there should be multiple camera positions, which the user can switch among as the program runs. You must support the following three:

Top-Down View: The camera looks down at Tommy from a high vantage point. The vantage point depends on Tommy's location, but not on the direction he is currently facing. For example, the camera might be 30 units above the ground to the south-east of Tommy's current position.

Following (Third-Person) View: The camera is placed behind Tommy and slightly above. The camera's position depends both on Tommy's position and the direction he is facing. Thus, the camera always sees Tommy's back and repositions itself when Tommy changes direction.

First-Person View: The camera is placed at Tommy's eyes. Through the use of the mouse motion, it is possible to look up and down, left and right.

Character control: The manner in which Tommy's motion is controlled should be sensitive to the camera view. The meaning of moving right (when the 'd' key is hit) should be relative to the viewer. In the following and first-person views, this would be to Tommy's right, but in the top-down view, this would be relative to the user's view.

Lighting: Your program must make use of at least one light source to illuminate elements of your scene. You are not required to generate a shadows, but it is not a bad idea to produce a hint of a shadow (e.g., a dark polygon drawn on the ground under the object) as a hint of where the character is located.

Texture mapping: Your program must have at least one element of texture mapping. For example, this might take the form of a skybox surrounding your scene and/or a texture placed on the ground.

Simple skeletal animation: Tommy should consist of a few moving parts (e.g., body and legs) and his walking behavior should involve an animation of these parts.

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.

You should implement your program using basic OpenGL and GLUT. You may use models and textures that you have downloaded from the Internet. You may make use of small technical code fragments as well, but remember to cite all of your sources in your ReadMe file. You should not use sophisticated graphics engines, however, to handle things like physics or camera control, since a key part of this project is learning how to implement these features. If you are unsure about the use of a resource, feel free to check with me.

Second Quiz (Compressed Form)

This "quiz" is closed-book and closed-notes. You may use 2 sheets 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 50 points. Good luck!

Problem 1. (15 points; 2–5 points each) Short answer questions. Except where noted, explanations are not required, but may be given for partial credit.

(a) OpenGL uses a *local lighting model*. Assuming you are using the basic OpenGL lighting system (as opposed to writing your own shaders) which of the following statements is *true*? (Select all the apply.)

(i) It does not support *indirect lighting* (light reflecting off one surface to illuminate another)
(ii) It does not support *caustics* (light refracting through one surface to illuminate another)
(iii) It does not support *shadows* (one object blocking light from another)
(iv) It does not support *specularity* (that is, shininess)

(b) OpenGL can optionally perform *perspective correction* when doing texture mapping. Under which of the following circumstances would you find this option most useful (select one):

(i) The polygon being textured is *uniformly close* to the viewer
(ii) The polygon being textured is *uniformly far* from the viewer
(iii) The polygon being textured *stretches from being close to being far*

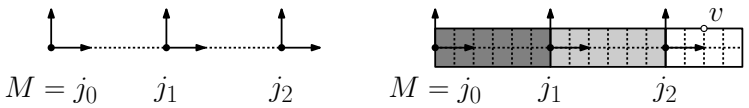
(c) Among the principal sources of latency in networked games (frame-rate latency, network protocol latency, transmission latency, and processing latency) over which does the game programmer have the greatest control? (Select one.) Briefly explain your answer.

(d) For each of the following (desirable) properties, which network protocol would be preferred, TCP or UDP:

(i) Improved reliability
(ii) Lower overhead
(iii) Maintaining order among packets
(iv) Improved transmission flow control

Problem 2. (10 points) The purpose of this problem is to derive the skinning transformation used in skeletal animation (just as we did in Lecture 15) from first principles.

Suppose you have a skeletal model consisting of three joints j_0, j_1, and j_2, where j_0 is the root of the model (see the figure below). Let $T_{[0\leftarrow 1]}$ denote the transformation that converts a point represented in j_1's coordinate frame to its representation in j_0's coordinate frame. Define $T_{[1\leftarrow 2]}$ analogously to convert from j_2's frame to j_1's frame. Also, define $T_{[1\leftarrow 0]}$ and $T_{[2\leftarrow 1]}$ to be the respective inverses of these transformations.

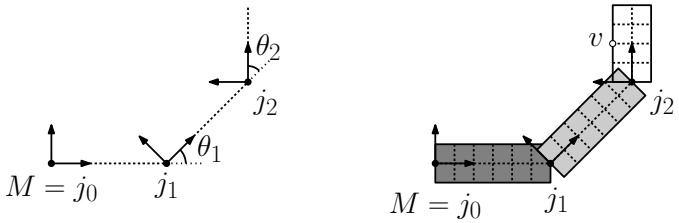


Problem 2(a)–(b).

(a) Given the above transformations and the fact that the model frame M is j_0, derive the *bind-pose transformation* $T_{M\leftarrow 2}$ that converts a point in j_2's coordinate frame to the model's frame.

(b) Given the above transformations, derive the *inverse bind-pose transformation* $T_{2\leftarrow M}$ that converts a point in the models frame to j_2's coordinate frame.

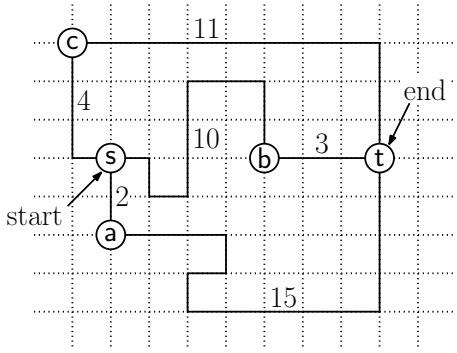
(c) Suppose that at time t, j_1 is rotated by angle θ_1, and j_2 is rotated by angle θ_2 (see the figure below). Let $R_1^{[t\leftarrow 0]}$ and $R_2^{[t\leftarrow 0]}$ denote the respective rotation transformations in local joint coordinates. Derive the *current pose transformation*, $T_{[M\leftarrow 2]}^{[t\leftarrow 0]}$, which maps a vertex in j_0's coordinate frame at time 0 to its position relative to the model frame at time t. Explain briefly. (In the lecture we also referred to this transformation as $C_{[M\leftarrow 2]}$.)



Problem 2(c)–(d).

(d) When the model is initially created in the bind pose, vertices are typically entered into the modeling software in *model coordinates*, not local joint coordinates. Derive the *skinning transformation* for joint j_2, which maps a vertex v bound to joint j_2 (but is given in model coordinates in the bind pose) to its model coordinates at time t. Explain briefly. (In the lecture we referred to this transformation as K_2.)

Problem 3. (10 points) In this problem we consider the execution of A* search on the graph shown in the figure below, where the path starts at s and ends at t. Each edge (u, v) is undirected and is labeled with its associated weight $w(u, v)$.

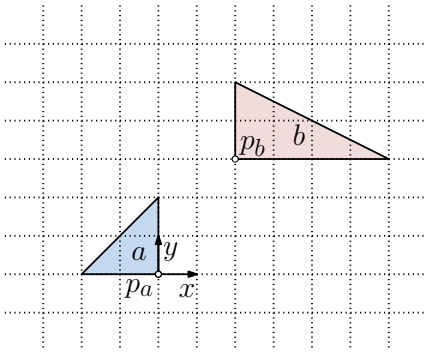


Problem 3.

As in Homework 2, the heuristic for A* search is the L_1 distance from the node to t. (Recall that this is sum of the absolute values of the difference of the x and y coordinates. For example, the L_1 distance between nodes a and t is $\text{dist}_1(a, t) = 7 + 2 = 9$, and therefore $h(a) = 9$.)

Trace the execution of A* search on this graph. Indicate the following: (1) list the nodes in the order in which they are processed, and (2) after processing each vertex indicate the d and f values of the nodes remaining to be processed. Recall that $f(u) = d[u] + h(u)$. (If there are ties for which node is to be processed next, select the node that is earliest in alphabetical order.)

Problem 4. (15 points) Consider the two right triangles a and b, shown in the figure below. (Triangle a's reference point is located at the origin, that is, $p_a = (0, 0)$ and b's reference point is at $p_b = (2, 3)$.)



Problem 4.

(a) Recall that *placing* object a at a point q means translating a so that p_a coincides with q. The *configuration obstacle* of b with respect to a is the set of placements of a so that it overlaps b. Describe (draw clearly or explain with coordinates) the configuration obstacle of b with respect to a.

(b) Recall that $\text{VO}_{a|b}$ is the set of all velocity vectors v such that, if triangle a moves with this velocity, it will intersect b at some time in the future, assuming that b is stationary. Describe $\text{VO}_{a|b}$ (again, draw clearly or explain with coordinates and slopes).

(c) Suppose that we are only interested in the time interval $\tau = [1, 4]$. (Note that the lower bound is *not* 0!) Define $\text{VO}^{\tau}_{a|b}$ to be the set of velocity vectors such that, if triangle a moves with this velocity, it will intersect b within the time interval τ. Describe this shape (again, draw clearly or explain with coordinates and slopes).