

Problem Set 2
CMSC 427
Distributed Tuesday, September 25, 2007
Due: Tuesday, October 9, 2007

Collisions, Intersections, Perspective and More Transformations

Programming

An executable for the finished version of this problem set is available on the class web page. You can obtain code that satisfies the conditions of problem set 1 by sending email to djacobs@cs.umd.edu (I'd prefer not to post it on the web). You can use this as a starting point, or you can use your own implementation of PS 1 to start. This code, and perhaps yours, handles the somersault. However, in this problem set, you don't need to worry about getting everything to interact properly with the somersault. Do not worry about maintaining proper function in the somersault as you add other features.

I have added a few features to the system that you are not required to implement. These include some print statements to indicate speed and intersections, and keyboard functions for 'd', 'e', 'r', and 't'. I found these helpful for debugging purposes and you might find similar functions useful.

Note that throughout the problem set you can assume that the rectanguloids in the scene are axial aligned, and do not intersect each other. The car will not usually be axial aligned, however. You can receive extra credit for handling more general rectanguloids in the scene.

By the way, it is certainly possible that my code has bugs. If it does, this does not excuse you if you have the same bugs. If you have any doubts about the program's desired performance, please ask.

1. **5 points.** Fix up the scene so that it looks nice, with a few rectanguloids scattered here and there. Make at least one rectanguloid tall, so that you can tip it over (see part 5). Make the cubes displayed with solid sides, instead of wire frames (look at the command *glPolygonMode*).
2. **15 points.** Add to the code so that the viewing position cannot go inside any of the cubes. If you try to go inside a cube, you will just stop, unmoving, outside the cube. When you are trying to enter the cube, it will slowly turn yellow. Note that if you do part 4, this part need not be present in your final code. However, doing this problem is a good warm up for part 4, and you can get credit for this part even if you are unable to get 4 working.
3. **15 points.** Now, instead of considering the viewer as occupying a single point at the viewing position, imagine that you are driving a rectanguloid car. This rectanguloid is aligned with the viewing direction. So when you turn, the car turns. The car is always pointing in the same direction in which you are viewing, so the direction you move is the direction you look in. Display the outline of the

car as a wire-frame (see *glPolygonMode*). You can make the car any size you want, but the wireframe of it should be visible as you move around.

4. **20 points.** Now, instead of the viewing position not being allowed to intersect the cubes, the whole car is not allowed to intersect a cube. If any point in the car intersects any point in the cube, you should stop moving (as in (2), the rectanguloid will slowly turn yellow if you keep trying to intersect it).
5. **15 points.** If a rectanguloid is tall and thin it can be tipped over. Specifically, if its height is more than twice as large as its length or width, it is tippable. When you hit the cube on any side, it will tip over in the opposite direction. That means the edge that is on the ground plane, opposite to where you hit it, will be fixed, and the rectanguloid will slowly rotate 90 degrees about this edge.
 - a. You should have the rectanguloid lie on its side once you are done. Future collisions with the car should be based on its new position. However, you will get partial credit if the rectanguloid goes back to its old position after it falls.
 - b. You do not need to worry about what happens if the rectanguloid intersects other objects as it falls.
 - c. You do not need to worry about the falling rectanguloid intersecting the car, until after it is done falling.
6. **Extra Credit, up to 15 points:** Build a game out of this. You could make a game just using these tools. For example, you could create a track and time your speed around it, perhaps subtracting points for hitting obstacles. Or you could create a maze that contains something you need to find.

You will receive more credit if you add more features to the system to use in your game. For example:

- When the car hits a rectanguloid, you can push the rectanguloid into a new position. This might be used to reveal hidden corridors. Special credit if the cubes move in a physically realistic way, and/or are allowed to become non-axial.
- Allow one falling rectanguloid to tip over another when it hits it. Make a set of falling dominoes.
- Add moving objects to the scene. You might chase them in your game, or have them chase you.
- Make up your own features!

If you do this part, please include a brief written description documenting your game, so that we can fully appreciate it.

Pencil and Paper Exercises

1. Perspective (2 points each)

- Suppose we have a perspective camera with a focal point at the origin. The image plane is the $z=1$ plane. We are looking at a point in the world located at $(-4, 2, 8)$. Where does this point appear in the image? (That is, give the 3D coordinates of a point in the image plane where this point will appear).
- Suppose the image plane were $z=2$. Now where would the point appear in the image?
- Suppose we have a perspective camera, with a focal point at the origin, and an image plane at the $z=1$ plane. We are looking at lines on the floor, which is the $y = -5$ plane. Give equations that describe the set of lines that have a vanishing point at $(3,0,1)$.

2. Projective Transformations (2 points each)

- Suppose we apply the following projective transformation to a set of planar points:

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Give the coordinates of a point in the resulting image that is on the horizon (eg., parallel lines in the original plane will appear to meet at this point in the transformed plane).

- Give the equation for the horizon line in the transformed plane.
- Does this perspective transformation correspond to a true perspective projection? That is, is the resulting image a possible perspective view of the original planar points? Why or why not?

3. Intersection (1 point): Consider a line segment with end points at $(12,6)$ and $(24,12)$.

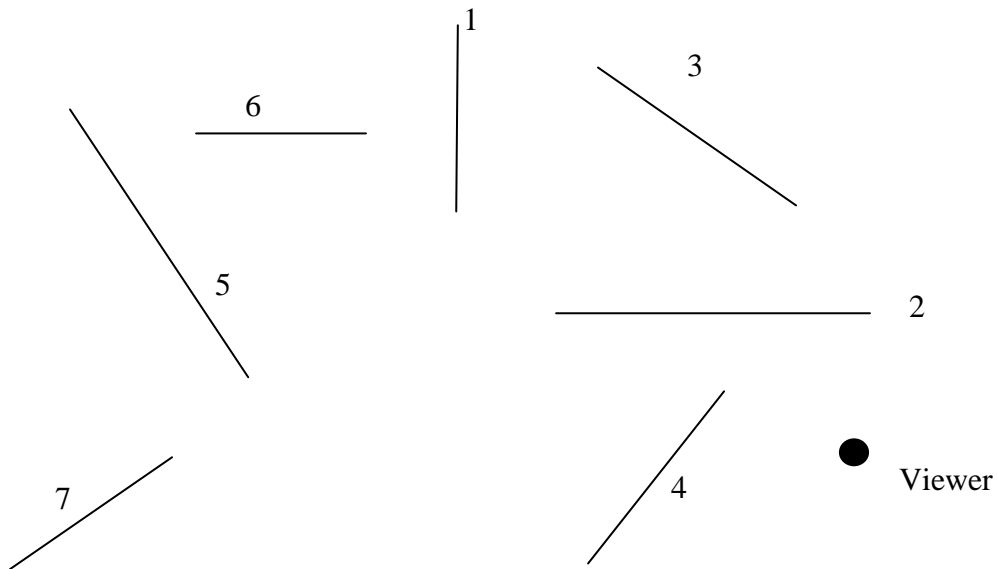
- Write an equation for the line that contains this line segment.
- Give a unit vector, n , that is perpendicular to this line segment.
- Use this to determine whether the points $(3,7)$ and $(30,18)$ are on the same side of this line or on opposite sides.
- Consider two triangles. The first has vertices at $(0,0)$, $(5,0)$ and $(0,5)$ and the second has vertices at $(4,2)$, $(8, 2)$, $(4,5)$. Do these triangles intersect?
- Suppose we put axial rectangles around the triangles in (e). Do these intersect?
- Consider the 3D triangle with vertices $(5, 7, 5)$, $(5, 11, 5)$, $(7, 9, 7)$. Give a unit vector for the normal to the plane that contains this triangle.
- Now consider a line that goes through the origin and through the point $(1, 3/2, 1)$. Where does this line intersect the plane in (f)?
- Is this intersection point inside the triangle of (f)?

4. Projection (4 points) Suppose we use the command `glOrtho(-3, 2, -1, 3, 2, 4)`. Give a 4x4 matrix that will transform points in the world so that any point that is

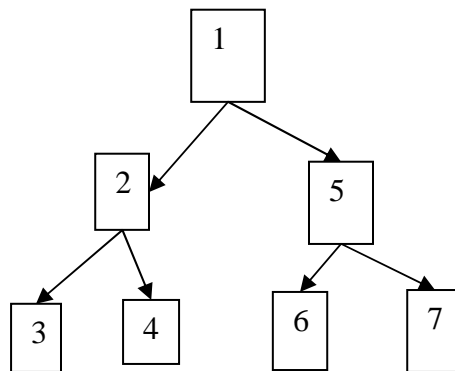
visible will be transformed so that it is in the square $-1 \leq x \leq 1$, $-1 \leq y \leq 1$, $-1 \leq z \leq 1$.

5. Visibility (3 points each)

a. Consider the 2D scene:



Here is a possible BSP tree for this scene:



Explain how this BSP tree can be used to choose the order in which we will render these line segments, from the point of view of the viewer shown in the figure. Give the order in which they will be rendered.

b. Construct a BSP tree for this scene in which 5 is at the root.