Computer Graphics - “Treasure Hunter”

CS 4830 — Dr. Mihail

February 25, 2019

1 Introduction

In this assignment you will implement an old technique to simulate 3D scenes called “billboarding”, sometimes referred to as “2.5D graphics” using sprites. A sprite is simply a scaled image that is always facing the camera. In other words, regardless of which direction the viewer is looking from, the sprite will be parallel to the view plane. Older games relied on this technique to simulate 3D objects to save computation time, which was required for real-time rendering, especially in games.

At a minimum, you will implement a game, which I uninterestingly call “Treasure Hunt”, where the goal is to navigate through a neighborhood that hides a treasure between many identical houses. The story, goal of the game and graphics are subject to creative enhancement, which I reward with extra credit. During game-play, the canvas should look (at a minimum) like Figure 2.

2 Implementation Details

The concept you have to master is the “camera”. The basic idea is that moving the camera (or player as s/he changes the way they look at the world) can be simulated by moving the world around the camera. Mathematically, the results are identical to creating a different projection matrix, but significantly more useful for 3D computer graphics because we know how to derive a perspective projection matrix whose center of projection is on the positive Z-axis, at some focal length d.

In this assignment, you will generate a set of objects that “live” in some 3D space, more specifically on the x-z plane. The player can move on that plane and is bound to it. The movement the player can make is forward and backward, and turn left or right. You will simulate the player moving through the world by transforming the object locations to simulate the player movement (i.e., the camera).

2.1 Ground and Sky

The sky and ground are each made up of two triangles, and drawn using a different shader. The skyground shader uses the color attribute, while the billboard shader uses the texture coordinate attribute. You have to disable depth testing, draw sky and ground, then enable depth testing and draw everything else, in that order.

2.2 Camera

When the player moves in the world, the camera is simulated by transforming every object position in the world. For example, if the player walks forward to an object, the camera moves closer to the object. Equivalently, we translate (move) the object closer to the camera. Similarly, when the player turns, the

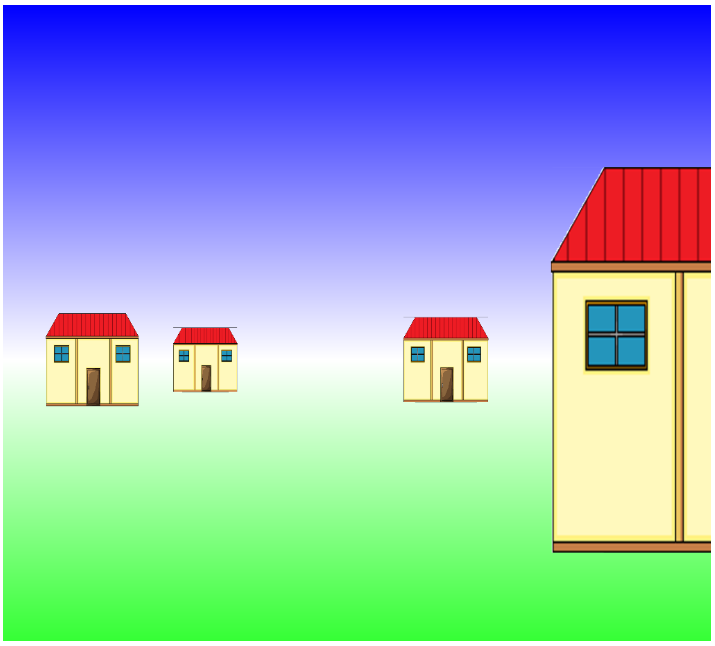

Figure 1: Sample canvas view during game-play

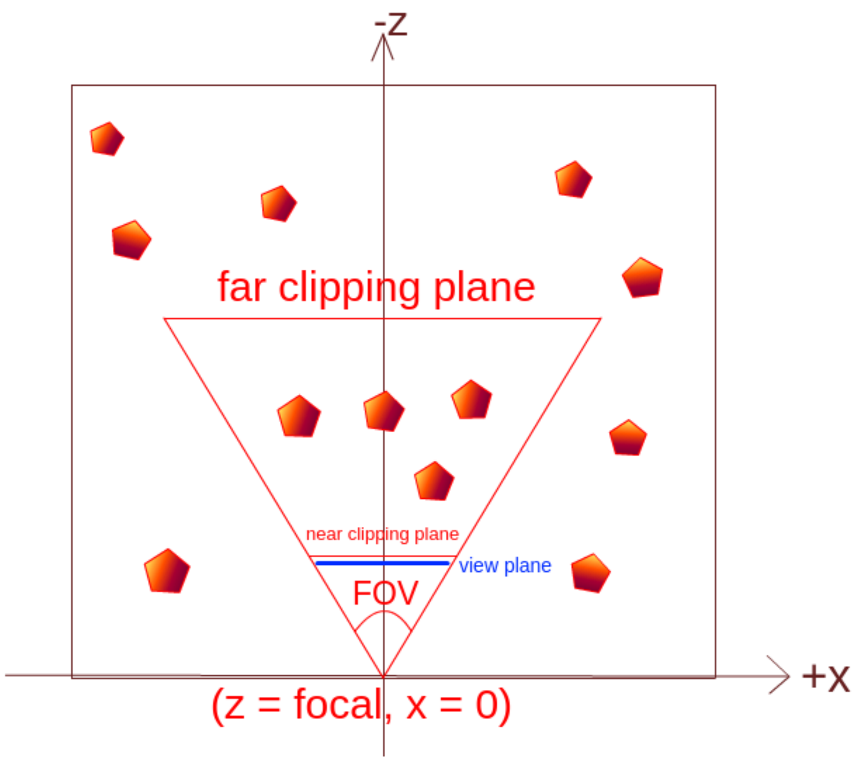

Figure 2: World coordinate system and camera. Objects within the near and far clip planes and within the field of view will be displayed on the canvas through a perspective projection onto the view plane at $z = focal$. The pentagons are the objects which are rendered as billboards using different textures (e.g., house or enemy).

direction the camera rotates. Equivalently, we can rotate every vertex of our objects in the world around the Y-axis, since our view plane is at $z = 0$.

You will implement two movements: forward/backward and turning left/right. Movement forward is equivalent to translating every vertex by a small positive amount along the Z-axis; movement backward is equivalent to a small negative translation along the Z-axis. Turning left is equivalent to rotating every vertex by a small degree in a clockwise direction. Turning right is equivalent to rotating every vertex by a small degree in a counter-clockwise direction.

2.3 Object collision

You can implement a simple collision detection algorithm by computing the distance from the origin of the world coordinate system to the object (the vector l2 norm) and considering a collision when that distance is below a threshold.

3 Program Requirements

There is no skeleton provided to you for this program. You may use any choice of linear algebra libraries (for matrix multiplication) or implement your own.

Your program has to follow the following specifications:

- Your program has to allow the player to move through the world with at least four controls: forward, backward, turn left and turn right.
- Your program has to use the context.drawImage() method for all the billboards.
- Your world has to contain billboards with at least two different images.
- Your world has to have at least 20 billboards. The house billboard is posted on the course webpage.

4 Technical Considerations

4.1 Projection

This program only simulates 3D geometry by drawing an image that is scaled with respect to the distance from the observer (camera). The perspective effect using this technique is achieved by scaling the image as a function of the object's distance along the z-axis from the camera. This is implemented by setting the image width and length as a float literal (e.g., 0.5) divided by the object's z coordinate in the world. Alternatively you can divide by the Euclidean distance, however, that causes the "fish-eye" effect in applications such as this.

When drawing the images on the canvas, the y coordinate will always be 0, since we're constraining our world to live on the x-z plane. The x coordinate of the projected images is computed by simply dividing the 3D x coordinate by z multiplied by a constant (that controls the field of view).

4.2 Drawing Images

When you draw two images in the same location using context.drawImage(), the image drawn with the last call will be visible. This means you have to implement a primitive form of depth testing such that if two objects are in a line of sight, the one closest to the camera is visible.

A simple way to achieve this is by sorting the objects by their z coordinate, in descending order and making calls to context.drawImage() in that order. If you choose to work with Javascript 2D arrays, the Array object has a .sort() method. It is important that you sort by the z-value and keep each tuple unchanged. The JS sort method can use a custom comparison method as follows:

```
data.sort(function(a, b){
    return a[0][2] - b[0][2]
});
```

The code above sorts a matrix in ascending order by the third column.

4.3 Game loop

The render loop will have the following logic (or something similar):

1. Clear canvas
2. Draw ground and sky
3. Draw objects (in reverse order of their z-values)
4. If W is pressed, increase the objects' z-values by a small amount
5. If S is pressed, decrease the objects' z-values by a small amount
6. If A is pressed, include a small clockwise rotation around the Y-axis
7. If D is pressed, include a small counter-clockwise rotation around the Y-axis
8. Do collision checking and/or other game logic

5 Due Date

This assignment is submitted via the electronic submission form on the course web page (as a zip file) and is due before midnight on Sunday, March 3rd.

6 Grading Rubric

- Your code has to draw at least some billboards before it gets considered for grading.
- W and S keys move the camera forward/backward (by translating the positions of objects along the Z-axis) (20 pts)
- A and D keys rotate the camera left and right (by rotating the positions of objects clockwise and counter-clockwise) (20 pts)
- Ground and sky are drawn (20 pts)
- Interaction with the world, e.g., finding an object when you get close to it (40 pts)

7 Extra Credit

It is possible to earn up to 100% extra credit. You can only get extra credit if the base requirements are met.

- Creative graphics, game logic and story-line (up to 25%)
- Billboard texture changes with respect to the angle it makes with the camera view-plane (up to 25%)
- “AI”s either chase you or run away from you (up to 25%)
- Multiple ways of interacting with the world (up to 25%)