Wet Chess - CS488 Final Project Manual

Arian Karbasi
a3karbas
20308751
November 6, 2011

Final Project:

Purpose: The purpose of this project is to create a two player chess game that incorporates various water graphics into it. The pieces will be modelled out of water, and a dead piece results in a puddle on the chess board.

Topics:

1. Texture Mapping

2. Modelling
3. Sound
4. Iterative Texture Mapping
5. Shadows
6. Shaders
7. Simulation of water
8. Perlin Noise

Statement:

This project will consist of a chess game with interesting water dynamics. The rules of chess will not change, but water graphics will be incorporated into the chess pieces. I modelled all of the chess pieces and make them look like they are made of water. An example of what this might look like is here at this link: http://www.youtube.com/watch?v=6sS6U7pTrvs. Each chess piece will have 'flowing' water and furthermore, each chess piece will drop a puddle of water after it is taken. The water pieces will be done using a perlin noise approach. Furthermore, the other side is done with the same approach except they will be made out of fire.

The pieces will all cast shadows and furthermore, ambient occlusion has been calculated for all of the pieces. This is demonstrated with a special ambient occlusion mode which has the capability to turn on ambient only lighting, to best demonstrate the self shadowing.

Hard shadows are calculated using a Shadow Mapping technique will be discussed later in the technical details.

Sound will also be incorporated into the game. The chess piece being taken will result in a 'splash' sound and the chess pieces moving will result in a sound that indicates the piece was moved. Finally, a sound will play to indicate that a checkmate has happened.

In addition, a skybox is implemented to give the background some life.

Milestones:

1) Read "Simplex Noise Demystified" paper. Pretty confused on how it works. Read a bunch on shaders. How do I debug this thing!?

2) Taking a break from shaders, started some architectural code for the chess game. Created the ChessGame, Piece, and Square class. Implemented simple move logic (can pieces go to a location) and implemented the make of the board (each position is a number from 0 to 63).

3) Today was an utter failure. I'm a complete n00b at Maya it turns out. I can't even select

objects...

4) Got texture mapping in the shader to work! That was pretty simple. Found a great bmp reader as well as a great shader class which handles all the monkey shader loading. But my textures don't map nicely, something is wrong with the texture coordinates.

5) Tried iterative texture mapping. It looks weird. I definitely need to figure out perlin noise.

6) Modelled the Queen, Pawn, Bishop, King, and the Rook. Each one is about 5000 polygons...I hope my things aren't super slow since I'll have 16 x 5000 faces. Also wrote a small custom obj reader to read it in to my custom mesh object.

7) Started implementing simplex noise. Apparently it looks nicer and its faster then perlin noise, might as well implement that.

8) Simplex noise still not working. Why the hell are shaders so hard to debug!?

9) Simplex noise working! But man it looks terrible. Oh man I hope things look good.

10) Played around with the parameters and numbers a little more, the noise looks a little better but still no where near water. I'll get back to it later.

11) Implemented the board. Consists of 64 individual scene nodes pointing to two meshes. I have my chess board makeup now and hooked up to the chess engine backend I made. Still need to do picking and proper chess logic.

12) So picking is done. I ended up giving each scene node a position so when I pick, I can pass in those positions to the backend and then it decides if it was a valid move (position is 0 to 63).

13) Simplex noise looks awesome now! I figured out my folley. I was using the noise in the wrong way, using it as a colour additive as opposed to a texture lookup offset. Changing it made it look way better.

14) Added noise to the vertices as well now. I have ripples now! Coupled with iterative texture mapping and my water looks so nice!

15) Bill helped me find a nice lava texture, and I applied iterative texture mapping plus a miniscule noise additive. The lava looks awesome. The way I mapped the coordinates makes it looks like its stretching out like Lava would...kinda got lucky with that one.

16) Now rendering 70000+ polygons with no lag whatsoever. Shaders are awesome (despite my hate before).

17) AAAAHH I HATE SHADERS. Been trying for days to get Jacobi iteration on the GPU

working with no dice. I can't get it debugging and nothing working. I'm going to give up.

18) Shadow mapping done! That was so frustrating. Took me a full day. Couldn't get frame buffers working, and then had trouble applying transformations to the texture I use to map coordinates (thanks to fabiensgrad for the awesome tutorial!)

19) Working on ambient occlusion now. Alex told me this is easy, it doesn't look easy.

20) Dammit alex! His ambient occlusion makes no sense. Reading GPU Gems now to see if I can implement disk area approximation

21) Disk area approximation works! But it looks weird...I'll bend the normals to see if that helps. Also my startup is really slow now.

22) Added a second occlusion pass to reduce double shadowing (multiply the disk area by the previous occlusion value to get a new area). Works awesomely.

23) Crap, these calculated occlusion values are per vertex. How the hell am I going to pass those down in the shader...I can't do that per vertex. Too slow. Scott suggests vertex buffer objects. Time to learn that.

24) Vertex buffer objects working! Things are fast, and smooth.

25) Refined a ton of noise parameters, water looks great now.

26) Sound, skybox, and camera spline still left...ugh I don't want to these boring ones.

27) Skybox is giving me trouble, why the hell is taking me so long...

28) Skybox works, found a bunch online. It looks like a bunch of others found the same images. Oh well, I guess we're all going to have the same skybox.

29) Sound is done, found some decent sounds. Decided to add background music.

30) Holy shit, modelling the knight is really hard. Failing really hard at it. Damn you Maya!

31) Added a WHOLE ton features and options. Fixed chess logic, added ripple options, flow options, lava options, shadowing options, added a luigi model to better demonstrate ambient occlusion, multiple skyboxes, multiple board textures, and position, game, and camera modes.

32) I'm failing really hard at the jacobi iteration. I think I'm not going to get it done. I can do the puddles another way.

33) Implemented puddles using a small mesh with deforms. It actually looks good. Can only

have one puddle at a time. Too tired and lazy to make more...

34) Finally added a sound for checkmate. Its a moo...I honestly couldn't find a better sound.

35) Awesome! Maya spits out smooth meshed normals, smooth shading looks awesome

36) Finally done! Just need to comment a refactor a whack ton since my code is terribly unorganized right now. I still need to add feedback for actually clicking pieces...but I might not have time for it. I really wanted to do refraction on the pieces, I'm definitely going to be working on this project past the deadline

User Interface:

- **Mode:** The game has 3 modes:
    - Position/Orientation: Rotate and position the board. B1 Translates in x and y. B2 translates in z. And B3 rotates the board.
    - Camera Rotate: Rotate the camera, only here to show off the skybox. Use B1 to rotate.
    - Game Mode: This is the mode where you actually play chess. When its your turn, click on the piece you wish to move, and then the next click should be the square you wish to move to.

- **Board Textures:** Simply change the board textures. Theres wood, metallic, and marble.
- **Lava Flow Speed:** Changes the speed at which the lava flows down the lava pieces
- **Water Flow Speed:** Changes the speed at which the water flows down the water pieces
- **Water Ripple Settings:** Two editables in here. We can edit the ripple speed or the ripple size. The difference is very clear in the water right away
- **Shadow:** Toggles shadows on or off. Didn't have time for nice soft shadows.
- **Occlusion:** Three occlusion settings:
    - Toggle Occlusion: Toggles ambient occlusion on or off. Turns off the water textures and makes the pieces white/black, to demonstrate ambient occlusion.
    - Lighting off: Turn on/off specular and diffuse lighting while in ambient mode. I did this to show that self shadowing is actually happening. The objects looks nice and 3d with only ambient lighting, shows that its working
    - Toggle Luigi: Just loads a luigi model I found online. I did this only because it best demontrates the occlusion. The pieces don't have as many crevices to show off whereas luigi has armpits and under his hat

  Music: Change the background music. Just downloaded some nice classical and 8bit music as options

  Skybox: Change the skybox background. I have mountains, two sky images, and one space image.

New Lua Commands

I have overwritten the old lua gr.mesh call. I now just pass in an obj file and my obj reader parses it and creates the mesh with all the texture coordinates and all of the normals. The code then generates a vertex buffer objects for that mesh. It also creates a disk for each vertex (used for ambient occlusion, more in the technical details).

Implementation of Objectives

- Texture Mapping - The texture mapping involved a few things. The first thing I do is load in all of the textures I'm going to use throughout the lifetime of the project. I load the images using a bmp reader I found online. Credit for the reader goes to beloni on gamedev.net. After loading in, I pass in the texture as a sampler2d which is then used in my shader to sample a colour from the texture. The texture coordinates for that vertex are defined in my vertex buffer. They were generated in maya where I mapped a uv image on to a picture to get nice looking texture coordinates. Now, I don't simply sample the vertex position in the image, my sampling is based on time, and the x,y,z of that vertex (gives it a fluid water look, more on this in simple noise).

- Ambient Occlusion - The technique I use for ambient occlusion is called the disk area approximation. The basic technique is as follows: for every vertex I create a disk with a position, normal, and area. The position and the normal are the same as the vertex (though the normal gets bent as the pass happens). The area is calculated as the sum of the area of all the faces which share that vertex divided by the total number of neighbouring faces. I had to code a data structure that knows all the neighbouring faces of each vertex.

  Occlusion values for each vertex are calculated as follows: For each disk, we have an emitter and a receiver. The emitter will be all of the other disks and the receiver is the receiving disk. For every emitter (all the disk), we calculate the the amount of shadow as

  1 - (d * cos(thetaemitter) * max(1, 4cos(thetareceiver)) / sqrt((receiver.area / pie) + d^2).
  Note that I had to muk around with this formula to get it to look nice but it was more or less the same. The max term handles getting rid of emitters that don't lie in the hemisphere. The disk area approximation was written about in GPU gems.

  This wasn't enough to make it look good. We don't want objects that are already in shadow to emit more shadow to the receiver. This makes crevices too dark. So I do a second pass which adjusts emitter area based on the previous occlusion factors calculated in the first pass to get rid of double shadowing. We could do more passes to get rid of triple shadowing and so on...this gave a very pleasing effect.

- Modelling - All of my meshes were created in Maya. Since the chess pieces are

symmetrical I only needed to draw ¼ of the pieces and simply copy and mirror the other sides. I had to write an OBJ reader to read in the obj's that maya spit out and use that create all of my needed data structures (such as the disks for my ambient occlusion approximation). As a note, I did NOT model the luigi model or the knight model. Those were free objs I downloaded. The luigi was to better show off ambient occlusion and for the knight, I had a lot of trouble getting it to look decent in maya and I was running out of time, so I downloaded an obj for it.

- Shadow Mapping - Shadow mapping involves two renders. In the first render, I render the scene once from the point of view of the light and then save ONLY the depth information in a texture. I then transform the light pov by taking the modelview and projection matrix as well as a small bias to get rid of self shadowing and then load that into a texture for use in a shader. The shader will use the texture to map the coordinates to the shadowed coordinates and then compare depth values. If it's less, we know our object was occluded in the light pov draw and we should draw it darker. Credit goes to fabiensgard for a great tutorial on frame buffer use and simple shadow mapping.

- Perlin Noise - I actually decided to implement simplex_noise instead so I will go over the implementation details here. First and foremost, the high level idea here is we pick the simplest shape in N-space that can be repeated to fill the space. In 2D space for example, the simplest shape is triangles, and in three dimensions the shape is a skewed tetrahedron. In 4D the shape is some 4D shape that is squashed along its main diagonal.

  The idea here, is that we have as few corners as possible making it easier to interpolate along the edges. So, for any given point, our noise contributions happen only from each corner and thus we are not interpolating as much as Perlin noise. Finally, we need to decide which simplex we're in by skewing the input along the main diagonal of the shape
  and then just taking the integer part of each coordinate for each dimension, in my case 4.

  As suggested by Sephen Gustovvsons paper on simplex noise, a simplex lookup table is used which holds the stepping order of each dimension. It is saved in a texture and passed into the shader for quick lookup once we decide the magnitude ordering.

  All of this, I implement in the shader (otherwise a noise for every fragment and vertex would simply be impossible). I preload all of the textures and 4D hypercube coordinates. All of these coordinates are provided by Gustovsson in his paper (and they are the same as perlins I believe). The following are all the steps I take in the shader to calculate noise:
  a. Define the skew and unskew variables (I have NO idea how Perlin/Gustovssen came up with the numbers...)
  b. next we skew our 4 dimensions to figure out which simplice we're in and we

compare each of the combinations of the 4 dimension coordinates to arrive at a lookup for our table.

c. The index of the lookup is achieved by taking a contribution from each corner in our simplice and use the lookup value as our step to calculate our offsets
d. Once we have our offsets, we do a lookup in the permutation texture (again, this texture was provided was Perlin). We use the result of the permutation texture to sample a specific midpoint in our complex 4D shape to arrive at a gradient and then we mix this gradient to get our contribution for one corner. We do this again for each of the possible 5 corners in our simplice and finally add them up to get our noise contribution
e. Note: I multiply the contribution by some large constant, otherwise it doesn't look as good

Once we have our noise the possibilities are endless. It can be used for simple refraction by messing with normals, or in my case, we can use it as an offset in the texture lookup so we don't lookup in the same place. I simply pass in a time variable as one of the dimensions so our offset changes per vertex per frame. Another usage I have for it is nice looking ripples. I offset the vertex position by my noise once again to achieve a very satisfying ripple effect.

- Skybox - The skybox is very simple. I simply draw 6 quads around my camera and texture map them. I took box textures online and mapped it to the 6 faces. This was very simple to do and doesn't need much explanation. Only that I use one giant texture and then pick smart texture coordinates in order to avoid seems (though some still exist).

- Shaders - Shaders are key to my program. The only reason I can render 70000 polygons a frame with 0 lag is because I do virtually no math in C++, all of it is done in the shader. I do shadow calculations, two noise calculations per vertex and fragment, texture mapping, matrix transformations, lighting calculations, and ambient occlusion.

  I also avoid if statement in shaders by passing in a uniform variable which is 1 or 0 and I multiply my values by that variable and either that calculation is kept or wiped out, this way I avoid nearly all if statements.

Bibliography

Bunnell, Michael. GPU Gems. Web. 20 Nov. 2011. <http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter14.html>.

Gustavson, Stefan. "Simplex Noise Demystified." Thesis. Linköping University, 2005. 2005. Web. 22 Nov. 2011. <http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf>.

"ShadowMapping with GLUT and GLSL." *Shadow Mapping*. Web. 19 Nov. 2011. <http://fabiensanglard.net/shadowmapping/index.php>.

Future Considerations:

- Use a library to get a good Chess AI as well as less bugs. My chess logic doesn't always work, especially in checkmates and mates where a piece can block the check.
- Make better puddles. I feel terrible I couldn't implement my jacobi iteration on the GPU and feel my game has a lot to gain from that
- Publish the code online, I wrote and learned a lot of things. Lots of which I didn't find good documentation or examples online.
- Refraction/Reflection. These shouldn't be too hard now that I'm comfortable with multi pass rendering and a more robust pipeline (shaders are awesome)
- Heat Haze. Can't be too hard, same concept as refraction really.
- Point light on the king. Will require 4 passes, one in each direction, but its doable and it'll look really nice

Acknowledgements:

There are a lot of people that made this project possible. First I'd like to thank all of the people in the lab who would help out whenever I got stuck. Specifically Razvan Vlaicu, Sean Hsueh, Bill He, Chris Xu, Jordan Haynes, and especially Scott Mckee for taking time off of his own projects and assignments to help us. All in all, I won't forget the countless hours spent in the lab.