6.111 Final Report - 3D City

Grace Cassidy and Khalil Elbaggari

09 December 2015

Table of Contents

1. Introduction

The project aims to render a 3D city as an immersive map, to serve as entertainment for a user to explore an unknown world. This virtual city is composed of wireframes buildings, and the user moves from a first person perspective. The user's movement is controlled with a GameCube controller. The city is implemented with wire frame buildings, and background sound was added to create a more dramatic effect. This project involves saving the frames to SRAM and accessing the frames with ZBT to write and read quickly. Additionally, the project involves computing the transformations to render the 3D images in 2D. The transformations are computationally intensive and require four multiplies, a divide, and trigonometric calculations for the building angles. The sound involves storing .coe converted from .wav files in the BRAM and later sending the appropriate sounds to the AC97 chip.

2. Overview

The project was designed to have several, increasingly complicated stages of implementation which improve the user's experience by making the virtual city more realistic. Breaking the project into stages allowed us to have working modules to demonstrate during the checkoff regardless of whether or not we were able to completely finish and integrate the entire project. The high level block diagram describing the organization of the project is shown in Figure 1. The foundational blocks of the project include reading and writing to RAM using ZBT to display a wireframe city on the VGA display. In this basic implementation, the user should be able to navigate throughout the city. The movement throughout the city is controlled by the buttons on the labkit or a GameCube controller to provide for a more natural user experience.

After implementing the virtual wireframe city, the next stage of the project involved adding audio. Various sounds found online in .wav files were converted to .coe files and then stored in the labkit's BRAM. The sounds are used for background footsteps as the user navigates throughout the virtual city, and sounds for the user

colliding with buildings. The background sounds improve the user experience and make the virtual city appear more realistic.

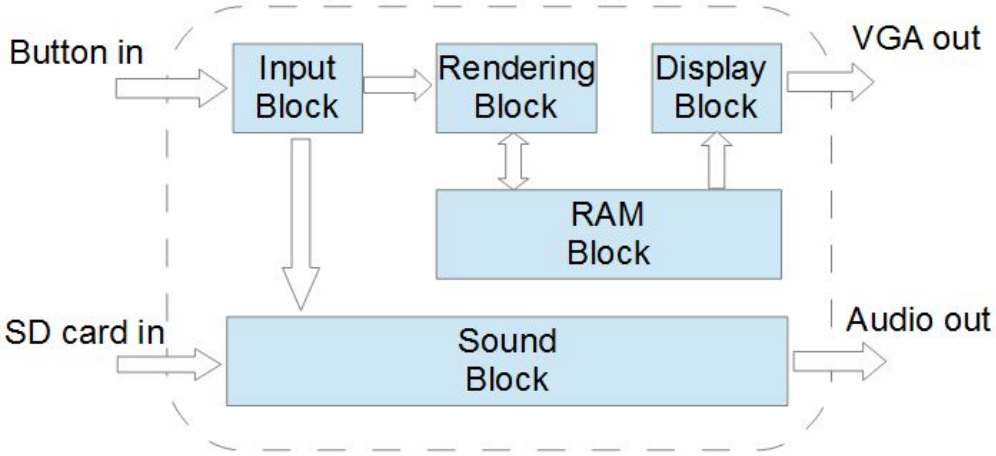

Fig. 1. High-Level Block Diagram

If we were to continue the project, the next stage of the project would involve replacing the wireframes in the imagined city with polygons. This stage will make the city look more realistic, but the polygons will also allow the project to eventually include shading of buildings to give the effect of a sun or other light source.

The next goal of this project is to implement physical laws into the game, such as acceleration of the user and velocity control. This would allow the user to feel like they are actually walking, running, or jumping when navigating the city.

In addition to the 6.111 labkit, this project interfaced with a GameCube controller. The controller was used to switch the navigation control from labkit buttons to a more natural control method. The GameCube controller and socket were bought online.

3. Implementation

The project was implemented on the 6.111 labkit. The labkit has known, working modules for ZBT SRAM unlike the Nexys 4 board. The Nexys has more resources, so it would be possible to render more complex scenes at a reasonable frame rate. All switches used were synchronized using the 6.111 synchronizer module, and all buttons used were debounced using the 6.111 debounce module.

3.1 Input Block (Khalil)

In order to move the user through this virtual city, the buttons on the labkit will modify the user's location in the city. The buttons will move side to side (left and right buttons) or translate (up and down buttons) the user, trigger events in the virtual city, or reset the user's position and locations of all objects. These locations and positions will be passed to the rendering module to be processed for eventual display on the VGA monitor. One of the last stretch goals will be using a GameCube controller to move the user through the virtual city and make movements more natural. This will also allow for more varied actions such as looking to the left while walking forwards. With the buttons, the user has to move in the direction they're facing.

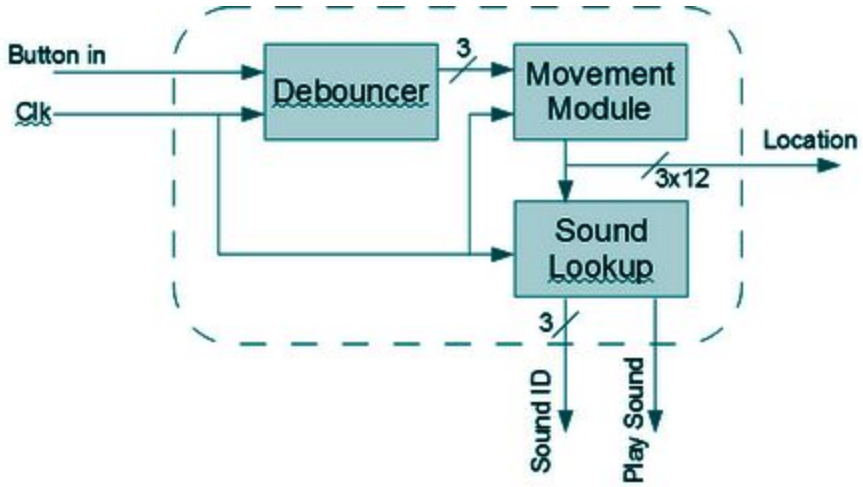

Fig. 2. Input Block Diagram

3.2 Rendering Block (Khalil)

The rendering block contains the rendering pipeline: edge module and matrix transformation. The edge module contains a counter to look up the IDs of the edges and their associated 8 bit vertices in 3D space. The number of bits, corresponding to spatial resolution, can be increased after testing. Since many of

the operations in the matrix transformation module are computationally expensive, thus slow, the module will be pipelined. The first part of the transform pipeline is computing cosines and sines from a fourth order, possibly fifth order, polynomial that approximate a quarter period of a cosine (fourth order) or sine (fifth order). The fourth order polynomial is easier computationally, but the fifth order has lower overall error. These calculations can be done in parallel and used in the transformation matrix. The transformation matrix requires three multiplications, and to generate the 2D projection on the VGA display, another multiply and divide are necessary. To keep the ID of the edge and transformed vertices associated, the ID needs to be passed down the rendering pipeline.

Once the rendering block is tested, constructing the buildings from polygons will make the virtual city appear more realistic. More computations are needed to implement the polygon faces, which may make the frame rendering slower. To implement the polygons, another point needs to be transformed from 3D to 2D, but the biggest computation will be determining the angle the surface forms with the light source.

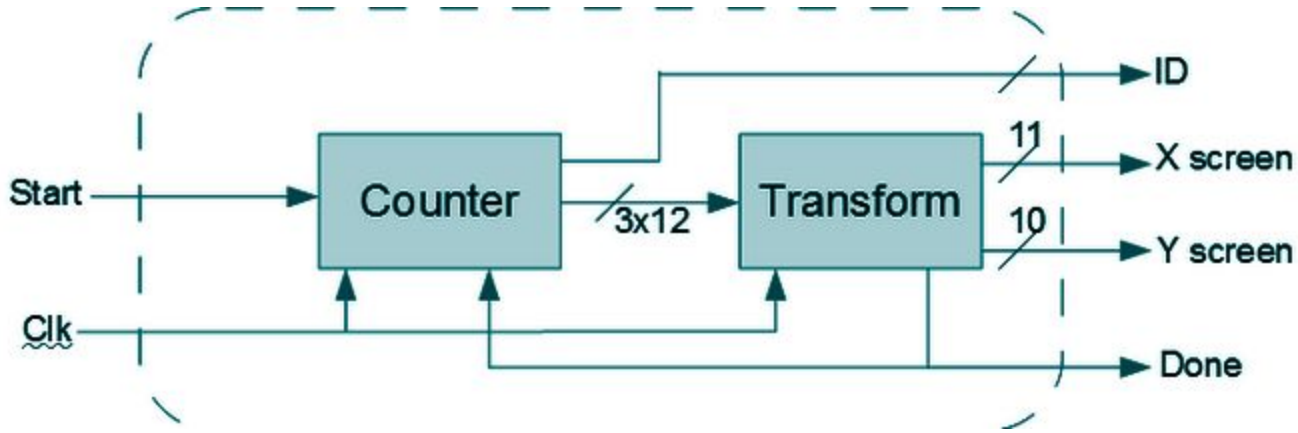

Fig. 3. Rendering Block Diagram

3.3 Line Calculations (Grace)

The line calculations are performed in the lines_1 module provided in the appendix. Before modules were integrated, a lookup table was used to send the x and y coordinates for the vertices in the images to be displayed. The lookup table sends the lines_1 module two vertices after each clock cycle.

The Bresenham line drawing algorithm is implemented to calculate the line between the two endpoints for each image. To increase the speed of this module, no multiplies or divides are needed to calculate the line. The difference in x0 and x1, and the difference in y0 in y1 are compared to determine the largest difference. Whichever direction had the largest pixel change determines in which direction the next pixel should be displayed in the line segment. This process is repeated until the module reaches the endpoint of the line segment, x1. To consistently move from x0 and x1, regardless of whether the x0 coordinate is further to the left of the x1 coordinate, the values are swapped if necessary to ensure that x0 to x1 is calculated from the left to the right of the screen.

3.4 Frame Buffer Module (Grace)

After the lines_1 module asserts lines_ready, meaning the wr_x and wr_y coordinates for an edge have been calculated by the lines_1 module, the 32-bit pixel value is written to one of the SRAM banks. The data for each address in SRAM is 36 bits wide, and the addresses are 19 bits wide. Since the pixel values were 32 bits, we were able to store one in each address. The wr_x and wr_y coordinates, the coordinates for the x and y values on the display, determine the address for the pixel value stored in memory. While one SRAM bank is being written to, the display module reads from the other SRAM bank. On the rising edge of the 65 MHz clock cycle, generated by the ramclock module, values are written to one SRAM, and values are read from the other SRAM bank to show on the XVGA display. When the vertical sync signal goes low, the banks are reversed: if the display were originally reading from bank 1 and writing to bank 0, after vsync goes low, the system would read from bank 0 and write to bank 1. To avoid ghost images from data stored previously, the data in the entire bank is erased with each switch of the frame buffers. Implementing the frame buffers allows the display to operate on a 65 MHz clock cycle and be displayed on the

XVGA display without flickering. Careful attention was paid to the timing of the reads and write for the SRAMs to implement ZBT. This required 2-stage pipelines of the signals.

To store the vertices of the buildings to be displayed on the VGA display, the RAM block will allow the system to read and write into the memory on the labkit. The RAM block includes three main modules. The first is the read module, the second is the write module, and the third is the scheduler to control the timing for these modules. These are separate modules to allow both partners to work on separate modules simultaneously. ZBT will be implemented within these blocks, and the sample ZBT code from the 6.111 website for the labkit will be used in the initial implementation.

3.5 Display Block (Grace)

The display_1 module determines whether the XVGA display should show a predetermined image, the white border or color bars, or if it should display the image read from SRAM. This block also generates the XVGA signals (hcount, vcount, hsync, vsync, and blank). The ramclock module provided by 6.111 provides the 65 MHz clock which is used for all of the modules in the display block.

To display the virtual city on an XVGA display as the user navigates throughout the city, a display block will be incorporated into the project. The display block will take the data read from the ZBT SRAM and will display the image on the monitor using 24-bit XVGA. Since accessing RAM takes longer than displaying the image on the display, there will be two frame buffers. One frame buffer will be used to display the current frame, and the other frame buffer will be used to read data from the RAM for the next frame to be displayed. We will try to implement the display block with a 65 MHz clock, but a slower clock, around 15-30 MHz, might be necessary to account for time to access the RAM.

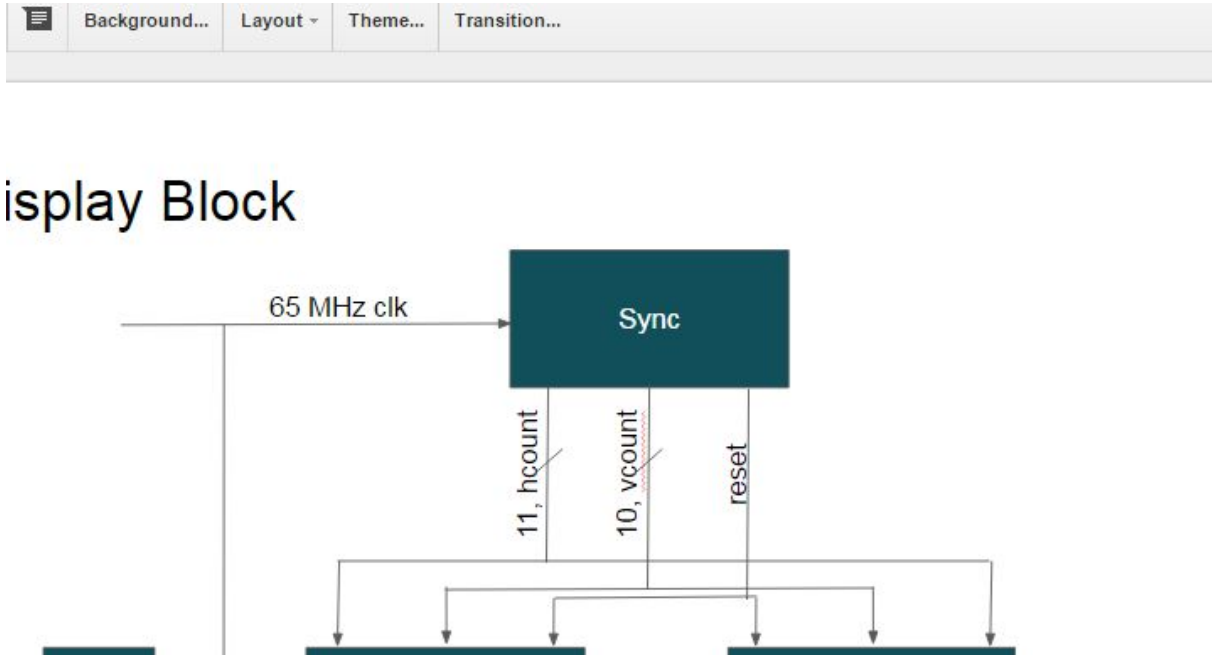

Fig. 4. Display and Frame Buffer Block Diagram

3.6 Audio Block (Grace)

In order to play sound while navigating through the virtual city, a sound block is implemented in the project. Sound samples, “Footsteps on Cement” and “Punch or Whack” originally in a .wav format from SoundBible.com were converted to .coe files using the provided MATLAB scripts. This required finding the sampling rate and converting it to 48 kHz if necessary to match the AC97’s sampling frequency. The sound files were sampled with 16 bits per sample to maintain high sound quality. Although this prevented long sound samples to be stored in memory due to a lack of storage space, this did not matter for our project because the sounds could be looped for the same effect. Two separate sound clips were stored in the labkit’s BRAM, both 16 bits by 64,000 bits. The “footstep” sound is constantly played while the user navigates throughout the virtual city, and the “whack” sound is play when the user navigates too close to a building and collides with a building or other object. These sounds were outputted to the headphone jack on the labkit after being sent to the AC97.

Since the input module was not integrated with the sound module, button 0 on the labkit controls the sound. If button 0 is depressed, the “footstep” sound stops playing, and the “whack” sound plays. The button 0 simulates the sound trigger from the input module.

Fig. 5. Sound Block Diagram

4. Testing

After completion of each module, the modules were tested with Verilog test benches to help reduce errors and aid in debugging. Test benches were also created to test groups of modules to further reduce debugging time. Testing each module individually made it easier to locate errors than waiting to debug until all modules were completed and integrated. There were some problems with testing the display and audio components of the project with test benches due to the interface with memory. Since test benches cannot simulate the memory in the labkit, the switches, buttons, and LEDs were more heavily relied on when testing these modules to determine different states modules were in, addresses modules were accessing in the memory, and other information to help debug the project. The XVGA display was also used to help debug the display modules such as the lines_1 module to verify the that the algorithm was operating correctly.

5. Conclusion and Recommendations

Overall, this project presented many challenges. Both the memory control and 3D to 2D transformations posed the greatest implementation obstacles. In addition to incorporating features learned in the previous 6.111 labs, the project also requires use

of ZBT SRAM and more intensive calculations such as multiplies and divides to complete the transformations. Implementation of these features furthered our knowledge of digital systems. The implementation involved navigating through a virtual, wireframe city and playing pre-recorded sounds when navigating through the virtual city.

One major bug was encountered when working on the final project. In the lines_1 module, x and y were of different bit widths, 11 bits and 10 bits, respectively, and switching x and y when doing the calculations caused errors in the final image. Originally, the lowest order bit was removed when moving x to y, but this did not work well. To combat this problem, x and y now both start out as 11 bits. The calculations are completed, and then when assigning the final x and y coordinates for the pixel locations in the line, only the top 10 high order bits are assigned to y to account for its 10 bit width.

While we were able to accomplish our commitment and goals, we did not have time to accomplish the stretch goals. The stretch goals included adding textures to the polygon surfaces, implementing shading of buildings, implementing a z-buffer to determine which surfaces of the building should be displayed, and adding acceleration to implement running or walking. The benefit of these stretch goals is to make the virtual city appear more realistic and improve the user's experience. The running and jumping features would have made the game more exciting by providing the user with options on how to explore the city. If someone another group were to continue working on this project, we would recommend starting the project earlier and ensuring that there is good communication between group members. This will make integration of modules easier, and it will save time towards the end of the project when the partners must work together to finish the project.

6. Sources

Fryer, Tim. "Footsteps on Cement." *SoundBible*. 13 Mar. 2013. Web. 1 Dec. 2015. <http://soundbible.com/2057-Footsteps-On-Cement.html>.

"Punch or Whack Sounds." *SoundBible*. 29 Nov. 2011. Web. 1 Dec. 2015. <http://soundbible.com/1952-Punch-Or-Whack.html>.

7. Appendix

The Verilog code for this project has been uploaded to the 6.111 course website.