Landscape Generator
CSEE W4840 Final Report

Calvin Hu
ch2880@columbia.edu
TA: Amandeep Chhabra

Table of Contents

1. Overview

The aim of this project is to create a hardware based system that generates landscapes and displays them on a monitor. There are also minimal controls using the switches and buttons to control rotation and color palette.

2. Algorithms Used

2.1 Diamond Square Algorithm

The Diamond-Square algorithm is used to generate the heightmap for the landscape. The method takes a n+1 by n+1 grid of height values. The four corners are seeded with values. The diamond step takes the average of the four corners and adds a random value and sets the center to that value. The square step takes the average of the new diamond corners created in the diamond step and takes their average and adds a random value and sets the center to that value. These two steps are repeated with smaller squares and diamonds until the grid is filled.

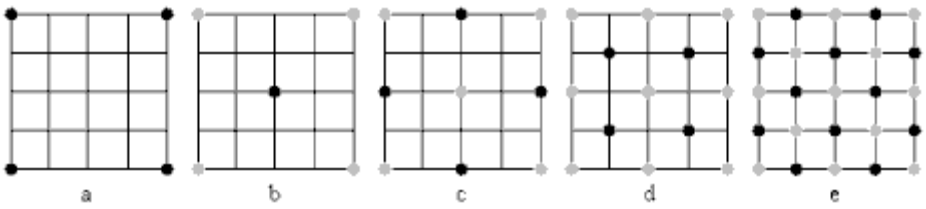

Figure 1. Diamond Square Algorithm diagram

2.2 Backface Culling Algorithm

The backface culling algorithm takes a face's surface normal and dot products it with the camera vector. If the dot product is greater or equal to zero, the face is not drawn.

2.3 Grid Sorting Algorithm

The height map is in a grid layout, so given the angle of rotation, the order in which each face is drawn can be known based on its distance from the camera vector.

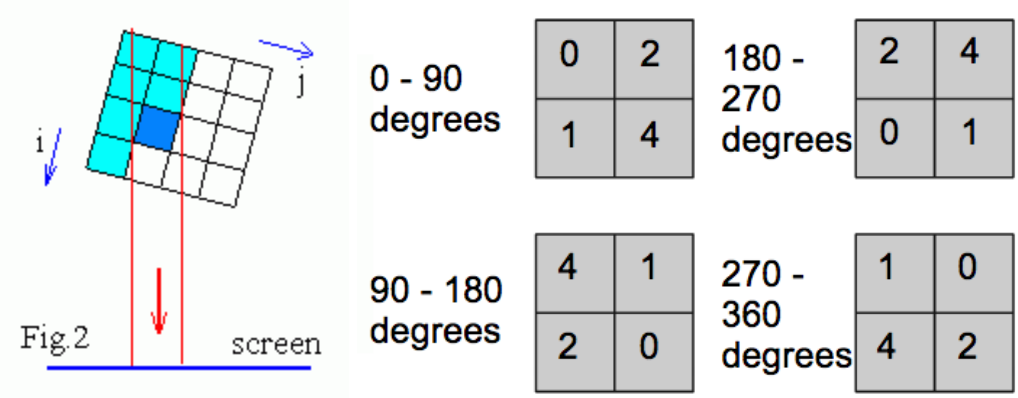

Figure 2. Grid sorting diagram

2.4 Rotation Algorithm

Representing a coordinate in 3D space as a 3 row column vector allows it to be transformed by multiplying with a rotation matrix.

$$\begin{bmatrix} \mathbf{d}_x \\ \mathbf{d}_y \\ \mathbf{d}_z \end{bmatrix} = \begin{bmatrix} \mathbf{a}_x \\ \mathbf{a}_y \\ \mathbf{a}_z \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Since only the x and y indices of the final transformed vertex are needed, we can reduce the computation to the following:

$$rotated\,x = x * cos(\theta) * cos(\varphi) + z * sin(\theta) - y * cos(\theta) * sin(\varphi)$$
$$rotated\,y = y * cos(\varphi) + x * cos(\varphi)$$

2.5 Bresenham's Line Algorithm

Bresenham's line algorithm maps an ideal line to screen coordinates, compensating for resolution. At every step, a pixel is drawn at the current x and y drawing coordinates, which are initialized with the starting coordinates. If the starting x coordinate is less than the ending x coordinate, swap the starting and ending points. If then the line has a positive slope, an error value is calculated by adding the slope to the error at every step. If the error value is less than or equal to 0.5, the current x drawing coordinate is incremented by 1, and the error value is incremented by the slope. Otherwise, if the error value is greater than 0.5, both the x and y drawing coordinates are incremented by 1, and the error value is incremented by the slope and decremented by 1. The drawing is completed when the current x and y coordinates match the coordinates of the end point. If the line has a negative slope, the same steps are followed, but the y value is decremented instead of incremented.

3. System Architecture

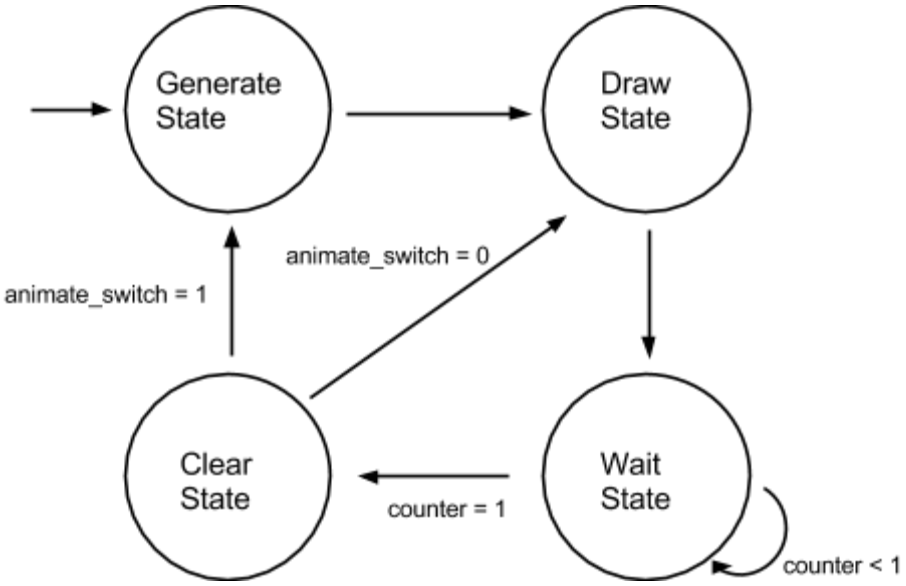

Figure 3. System State Machine

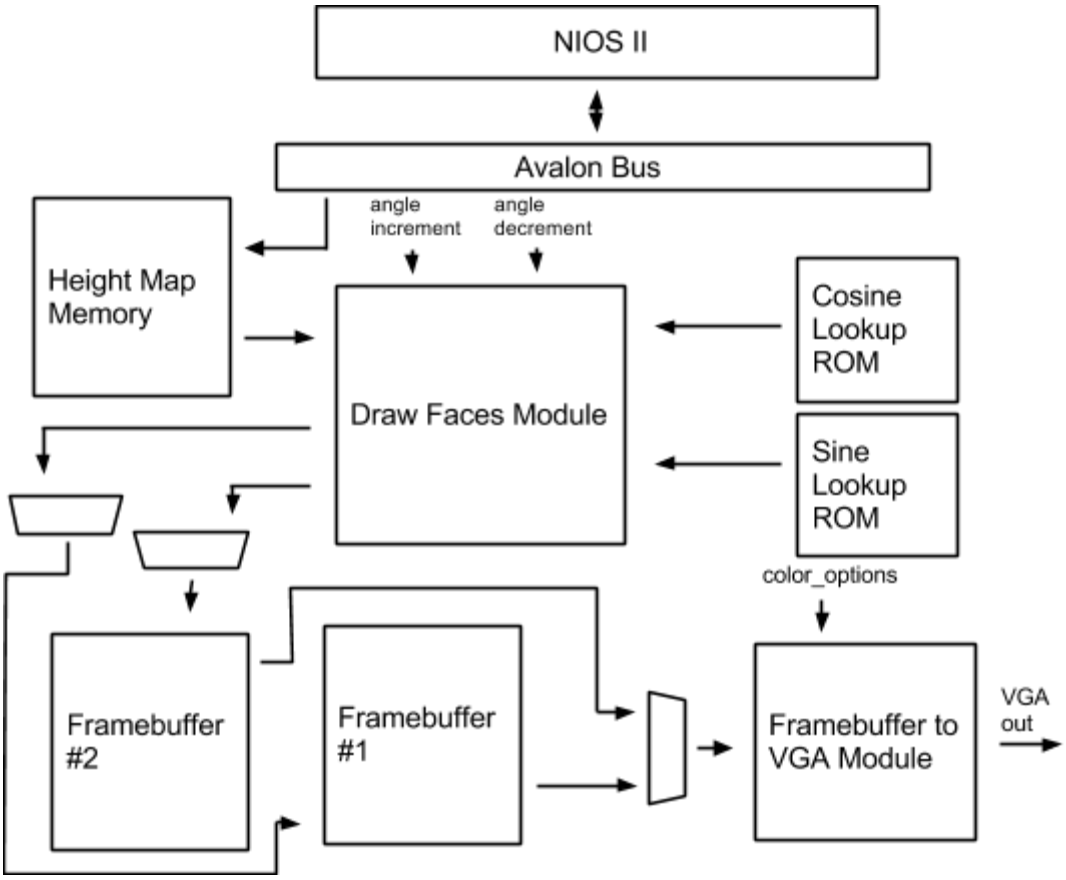

Figure 4. High Level Block Diagram of the System Architecture

- The framebuffer being read from and being written to are set and swapped every time a write to the framebuffer is finished.
- The framebuffer to VGA module is always running, reading from whichever framebuffer is set as the framebuffer to be read from. The framebuffers are only switched after a complete screen is drawn. The entire system uses a 25 MHz clock to avoid miscommunications with the VGA module.
- **Generate State:** The NIOS II processor generates the height map, and writes to height map RAM using a controller connected to the Avalon Bus.
- **Draw State:** The Draw Faces Module then reads the height map RAM and draws the converted image to the set framebuffer.
- **Wait State:** The system waits for the current frame to finish drawing before swapping the framebuffers.
- **Clear State:** The framebuffer to be written to is cleared (filled with "00"s). If the animate switch is on, the state machine will return to the generate state. Otherwise, it will return to the draw state. If rotate left or rotate right button is pressed, the signal to change the angle of the drawing is sent to the Draw Faces Module.

4. Design Implementation

4.1 Fixed Point Number system

The fixed point number system represents a number as a total of 36 digits. The first 18 digits are used to represent the integer portion of the number. The second 18 digits are used for the fractional portion of the number. The 36 bit length of the system was chosen because the embedded multipliers in the DE2 are 9 bits. To convert a standard value into this fixed point system, multiply the number by 2^18 or 262144, truncate the fraction portion and convert to base 2. For example, to convert 36.57, multiply by 262144 to get 9586606.08, truncate the .08, and convert 9586606 to 100100100100011110101110.

Integer	Fractional
35 to 18	17 to 0

4.2 Memory Allocation

4.2.1 Sine/Cosine Lookup ROM

These are two 180 x 20 bit block ROMs that act as sine and cosine lookup tables. Each row holds a 20 bit value that represents a sine or cosine value in fixed point notation. The address of the ROM acts as a degree value, where the data at address x is the sine/cosine value at degree x. Values from 0 degrees to 179 degrees are stored, the remaining values use inverted values. The ROM is initialized with values in a .mif file.

4.2.2 Height Map RAM

The height map is stored in a 1024 x 36 bit block RAM. The data is organized as a 2D grid where each set of 32 addresses forms a row, with 32 columns and 32 rows total. Each entry holds a 36 bit data value that represents a height in fixed point notation.

4.2.3 Frame Buffer RAMs

The two frame buffer RAMs used as a double buffer, each with a size of 76800 x 2 bits. Each each of 320 addresses forms a row of the frame with 240 rows total making a frame of 320 by 240 pixels.

4.3 Height Map Generation

The height map generation is handled by the NIOS II cpu. The values are written into floats in a 2D array in the software using the C rand() function and diamond-square algorithm. The values are then copied from the 2D array into the height map RAM using a controller connected to the Avalon bus. The values are written as 32 bit words into the controller, which then pads 4 bits before writing to the height map RAM.

4.4 Draw Faces Module

After the height map is generated and written to the height map RAM, the Draw Faces Module draws looks to see the Draw Faces Module looks at the current angle of the display and determines which one of the four grid sort orders the faces will be read in. The sine and cosine lookup tables are accessed to retrieve the values for the current angle. The heightmap is read in sets of 4 (a quadrilateral face), the top left, top right, bottom left, and bottom right. After each of these values are read, it is multiplied against the rotation matrix to find the new rotated x and rotated y indices. After all four rotated vertices have been calculated, the z component of the normal vector can be calculated and if it is less than 0, the face is backface culled and not drawn. Otherwise, the edges of the faces are drawn using the lines module.

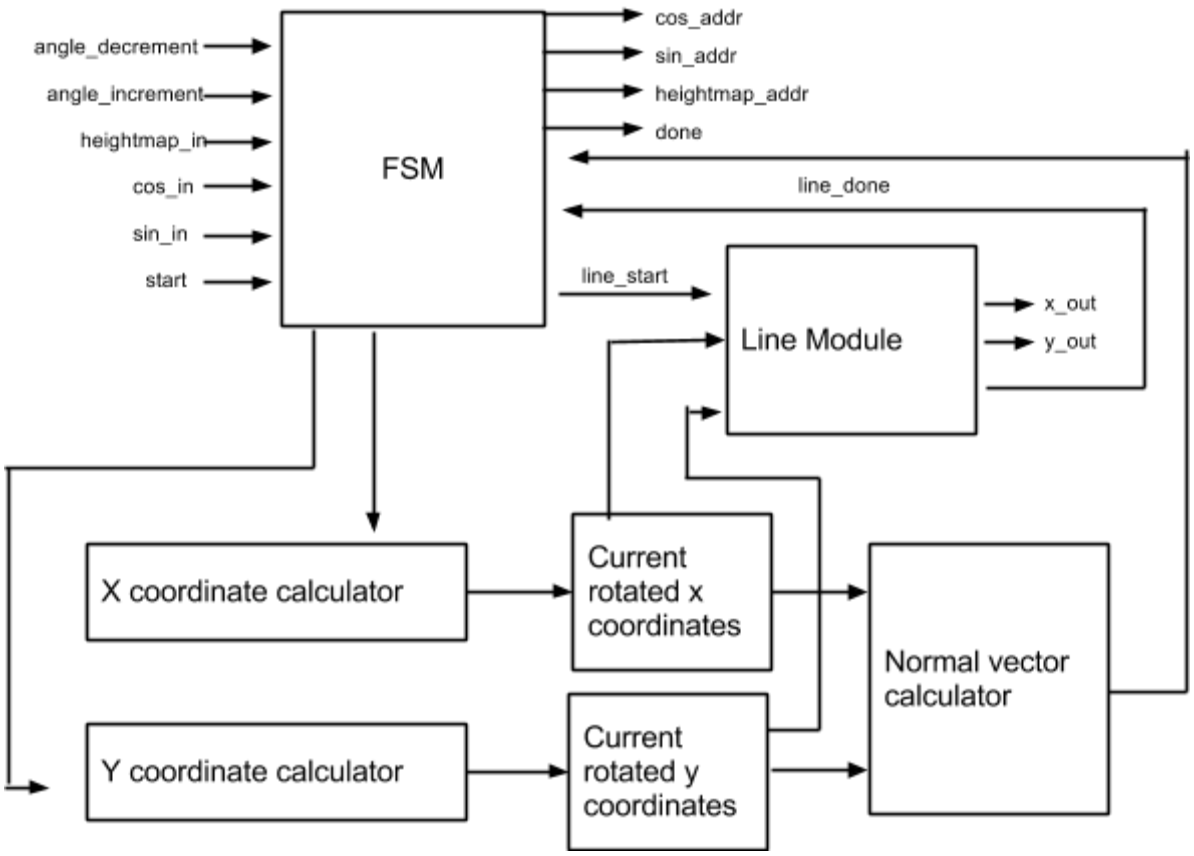

Figure 5. Draw Faces Module High Level Block Diagram

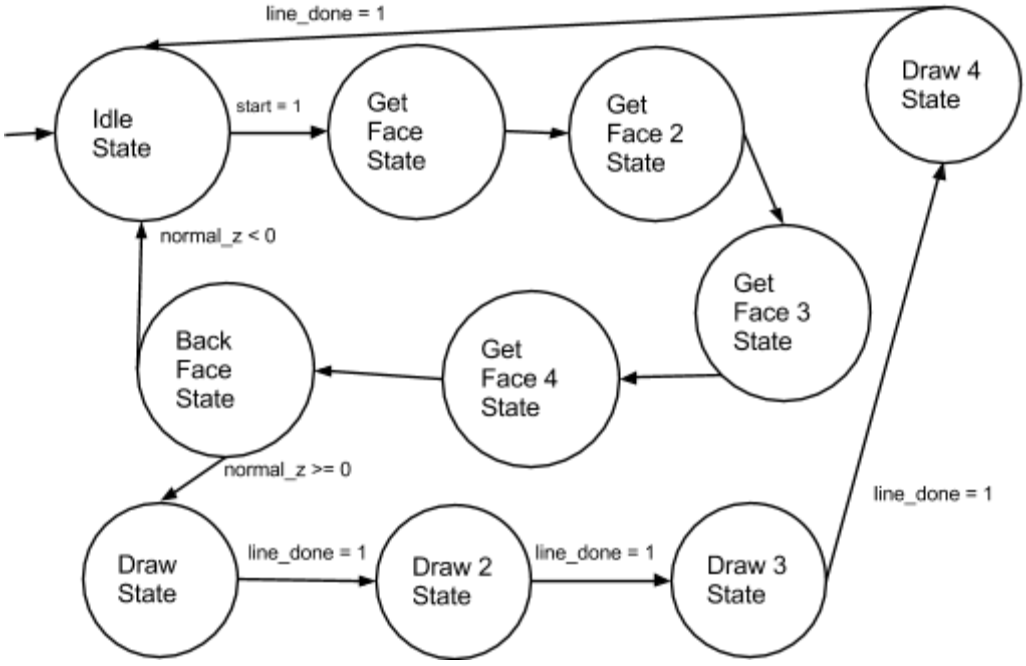

Figure 6. Draw Faces Module State Machine

4.5 Framebuffer To VGA Module

The Framebuffer to VGA Module reads from a framebuffer and outputs to a VGA connector. It is based on the lab 3 VGA output, and outputs different colors based on the color_options signal, which changes the color table the framebuffer values are mapped to. Because the frame buffer has a resolution of 320 by 240, but VGA has a resolution of 640 by 480, each pixel in the framebuffer is drawn as a 2x2 pixel size in VGA. Because the entire system runs at 25 MHz, it was not necessarily to use a different clock to run the system.

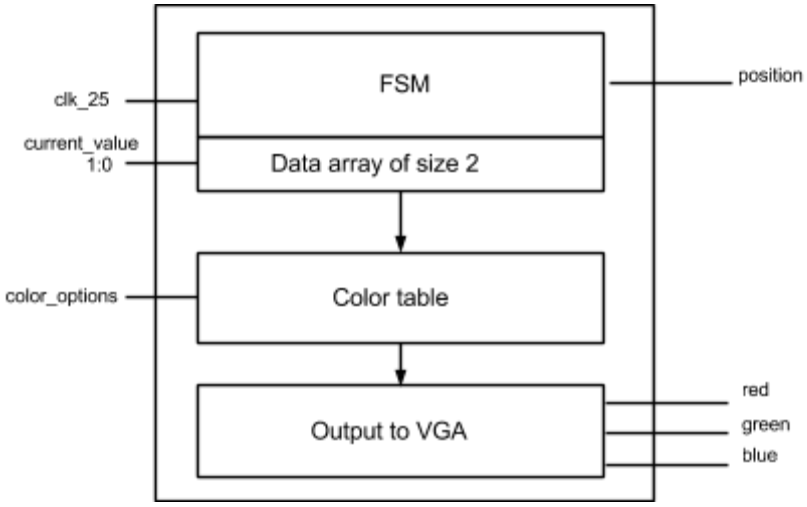

Figure 7. Framebuffer to VGA Module High Level Block Diagram

5. Design Changes

Memory changes

To save memory for two framebuffers, the framebuffer resolution had to be reduced from 640 by 480 pixels to 320 by 240 pixels. The sine and cosine lookup tables' widths had to be reduced from 36 bits to 20 bits and the depth had to be reduced from 360 addresses to 180. Originally, the heightmap memory was 108 by 1024, but was reduced down to 36 by 1024, because the x and y coordinates could be inferred from the read address.

Draw Face Module

Many components of the Draw Face Module were originally instead part of the top level design, but I found that this organization unnecessarily made too many reused signals external to each small module and mixed top level functionality with the drawing functionality in the top level file.

Scanline Fill Algorithm

Instead of using the Bresenham Line Algorithm module, a Scanline Fill Algorithm was planned. However, it wasn't completed in time, and was much slower than the Bresenham Line Algorithm.

6. Lessons Learned

This project showed me a lot about both what it means to handle a large assignment and working with VHDL and hardware. Working with animated graphics in hardware was tricky because of the large amount of values being thrown around, so working incrementally was very important. The lengthy compile times also had to be used wisely. It was necessary to consistently check new components with the existing system to make sure they worked well.

7. Source Code

7.1 VHDL

FractalLandscapeGeneratorTop.vhd

```
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity FractalLandscapeGeneratorTop is
	port(
		signal CLOCK_50 : in std_logic;
		signal LEDR : out std_logic_vector(17 downto 0);
		signal SW : in std_logic_vector(17 downto 0);
		signal KEY : in std_logic_vector(3 downto 0);

		SRAM_DQ : inout std_logic_vector(15 downto 0);
		SRAM_ADDR : out std_logic_vector(17 downto 0);
		SRAM_UB_N,
		SRAM_LB_N,
		SRAM_WE_N,
		SRAM_CE_N,
		SRAM_OE_N : out std_logic;


		VGA_CLK,
		VGA_HS,
		VGA_VS,
		VGA_BLANK,
		VGA_SYNC : out std_logic;
		VGA_R,
		VGA_G,
		VGA_B : out unsigned(9 downto 0)
	);
end FractalLandscapeGeneratorTop;

architecture rtl of FractalLandscapeGeneratorTop is

	signal clk25 : std_logic := '0';

	type states is (GEN_STATE, DRAWFACES_STATE, WAIT_STATE, CLEAR_STATE);
	signal state : states := DRAWFACES_STATE;

	signal drawfaces_start, drawfaces_done, frame_done : std_logic := '0';
	signal xpos, ypos : std_logic_vector(9 downto 0);
	signal position : std_logic_vector(18 downto 0);

	signal faces_rdaddress, faces_wraddress : STD_LOGIC_VECTOR (9 DOWNTO 0);
	signal faces_data : STD_LOGIC_VECTOR (35 DOWNTO 0);
	signal faces_wren   : STD_LOGIC := '0';
	signal faces_q      : STD_LOGIC_VECTOR (35 DOWNTO 0);
	signal backface_on : std_logic := '1';
	signal z_section : std_logic_vector(5 downto 0) := "111111";

	signal current_image_value, fbuffer1_data, fbuffer2_data, drawfaces_fdata :
```

```
STD_LOGIC_VECTOR (1 DOWNTO 0);
	signal fbuffer1_rdaddress, fbuffer2_rdaddress : STD_LOGIC_VECTOR (16 DOWNTO 0);
	signal fbuffer1_wraddress, fbuffer2_wraddress, drawfaces_faddress :
STD_LOGIC_VECTOR (16 DOWNTO 0);
	signal fbuffer1_wren, fbuffer2_wren, drawfaces_fwren : STD_LOGIC  := '0';
	signal fbuffer1_q, fbuffer2_q : STD_LOGIC_VECTOR (1 DOWNTO 0);

	signal fbuffer_bool : std_logic := '0';

	signal counter : integer := 0;
	signal clear_counter : integer := 0;

	signal gen_start, gen_done, gen_wren : std_logic := '0';
	signal draw_rdaddress, gen_rdaddress : std_logic_vector(9 downto 0);
	signal in_key : std_logic_vector(3 downto 0) := "1111";
	signal in_sw : std_logic_vector(17 downto 0) := (others => '1');

	signal reset_n : std_logic := '1';
	signal next_hpos, next_vpos : std_logic_vector(15 downto 0);
	signal line_done, line_start, line_reset, line_plot : std_logic := '0';
	signal line_data : std_logic_vector(0 downto 0) := "1";
	signal x1,x2,x3,x4,y1,y2,y3,y4 : std_logic_vector(10 downto 0);
	signal rotcontrol_z : std_logic_vector(1 downto 0) := "00";
	signal debounce_counter : integer := 0;
	signal color_options : std_logic_vector(2 downto 0) := "000";


begin
	process(CLOCK_50)
	begin
		if rising_edge(CLOCK_50) then
			clk25 <= not clk25;
		end if;

		if rising_edge(clk25) then
			case state is
			when GEN_STATE =>
				LEDR <= (0 => '1', others => '0');
				reset_n <= '1';
				if gen_done = '1' then
					state <= DRAWFACES_STATE;
					gen_start <= '0';
				else
					state <= GEN_STATE;
					gen_start <= '1';
				end if;
			when DRAWFACES_STATE =>
				LEDR <= (1 => '1', others => '0');
				reset_n <= '1';
```

```
                        if drawfaces_done = '1' then
                                state <= WAIT_STATE;
                                drawfaces_start <= '0';
                        else
                                state <= DRAWFACES_STATE;
                                drawfaces_start <= '1';
                        end if;
                when WAIT_STATE =>
                        LEDR <= (2 => '1', others => '0');
                        drawfaces_start <= '0';

                        if counter = 1 then
                                counter <= 0;
                                clear_counter <= 0;
                                state <= CLEAR_STATE;
                                fbuffer_bool <= not fbuffer_bool;
                        else
                                if frame_done = '1' then
                                        counter <= counter + 1;
                                end if;
                                state <= WAIT_STATE;
                        end if;

                when CLEAR_STATE =>
                        LEDR <= (3 => '1', others => '0');

                        if clear_counter < 76800 then
                                clear_counter <= clear_counter + 1;
                                state <= CLEAR_STATE;
                        else
                                if in_key(0) =  '0' then
                                        state <= GEN_STATE;
                                else
                                        state <= DRAWFACES_STATE;
                                end if;
                        end if;
                end case;

                if in_key(0) /= KEY(0) then
                        if debounce_counter >= 500 then
                                in_key(0) <= KEY(0);
                                debounce_counter <= 0;
                        else
                                debounce_counter <= debounce_counter + 1;
                        end if;
                end if;
                if in_key(1) /= KEY(1) then
                        if debounce_counter >= 500 then
                                in_key(1) <= KEY(1);
```

```
                                debounce_counter <= 0;
                        else
                                debounce_counter <= debounce_counter + 1;
                        end if;
                end if;
                if in_key(2) /= KEY(2) then
                        if debounce_counter >= 500 then
                                in_key(2) <= KEY(2);
                                debounce_counter <= 0;
                        else
                                debounce_counter <= debounce_counter + 1;
                        end if;
                end if;
                if in_sw /= SW then
                        if debounce_counter >= 500 then
                                in_sw <= SW;
                                debounce_counter <= 0;
                        else
                                debounce_counter <= debounce_counter + 1;
                        end if;
                end if;
                color_options <= in_sw(7 downto 5);
                rotcontrol_z(1) <= in_key(2);
                rotcontrol_z(0) <= in_key(1);
        end if;
    end process;

    faces_wren <= gen_wren when state = GEN_STATE else '0';
    faces_rdaddress <= gen_rdaddress when state = GEN_STATE else draw_rdaddress;

    current_image_value <= fbuffer1_q when fbuffer_bool = '0' else fbuffer2_q;
    fbuffer1_rdaddress <= position(16 downto 0);
    fbuffer1_wraddress <= std_logic_vector(to_unsigned(clear_counter, 17)) when
state = CLEAR_STATE else drawfaces_faddress;
    fbuffer1_data <= "00" when state = CLEAR_STATE else drawfaces_fdata;
    fbuffer1_wren <= '1' when state = CLEAR_STATE and fbuffer_bool = '1' else
                                        drawfaces_fwren when fbuffer_bool = '1'
else
                                        '0';

    fbuffer2_rdaddress <= position(16 downto 0);
    fbuffer2_wraddress <= std_logic_vector(to_unsigned(clear_counter, 17)) when
state = CLEAR_STATE else drawfaces_faddress;
    fbuffer2_data <= "00" when state = CLEAR_STATE else drawfaces_fdata;
    fbuffer2_wren <= '1' when state = CLEAR_STATE and fbuffer_bool = '0' else
                                        drawfaces_fwren when fbuffer_bool = '0'
else
                                            '0';
```

```
	image_to_vga : entity work.ImageToVGA port map(
		reset => '0',
		clk => clk25,
		FRAME_DONE => frame_done,
		VALUE_AT_CUR => current_image_value,
		--VALUE_AT_CUR => '1',
		XPOS => xpos,
		YPOS => ypos,
		position => position,
		color_options => color_options,
		VGA_CLK => VGA_CLK,
		VGA_HS => VGA_HS,
		VGA_VS => VGA_VS,
		VGA_BLANK => VGA_BLANK,
		VGA_SYNC => VGA_SYNC,
		VGA_G => VGA_G,
		VGA_B => VGA_B,
		VGA_R => VGA_R
	);

	frame_buffer1 : entity work.framebuffer_ram port map(
		rdaddress    => fbuffer1_rdaddress,
		wraddress    => fbuffer1_wraddress,
		clock => clk25,
		data   => fbuffer1_data,
		wren   => fbuffer1_wren,
		q      => fbuffer1_q

	);

	frame_buffer2 : entity work.framebuffer_ram port map(
		rdaddress    => fbuffer2_rdaddress,
		wraddress    => fbuffer2_wraddress,
		clock => clk25,
		data   => fbuffer2_data,
		wren   => fbuffer2_wren,
		q      => fbuffer2_q

	);

	drawfaces : entity work.DrawFacesModule port map(
		clk => clk25,
		start => drawfaces_start,
		done => drawfaces_done,

		rdaddress => draw_rdaddress,
		q => faces_q,

		frame_address => drawfaces_faddress,
```

```
              frame_data => drawfaces_fdata,
              frame_wren => drawfaces_fwren,
              backface_on => backface_on,
              z_section => z_section,
              rotcontrol_z => rotcontrol_z
       );

       hmap_memory : entity work.HeightMapMemory port map(
              rdaddress => faces_rdaddress,
              wraddress => faces_wraddress,
              clock  => clk25,
              data => faces_data,
              wren => faces_wren,
              q => faces_q
       );

       nios2 : entity work.nios_system port map(
         clk_0 => clk25,
         reset_n => reset_n,

     data_from_the_de2_cpu_to_vga_0 => faces_data,
     gen_done_from_the_de2_cpu_to_vga_0 => gen_done,
         gen_start_to_the_de2_cpu_to_vga_0 => gen_start,
         hmap_address_from_the_de2_cpu_to_vga_0 => faces_wraddress,
         wren_from_the_de2_cpu_to_vga_0 => gen_wren,

  -- the_sram
         SRAM_ADDR_from_the_sram => SRAM_ADDR,
         SRAM_CE_N_from_the_sram => SRAM_CE_N,
         SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
         SRAM_LB_N_from_the_sram => SRAM_LB_N,
         SRAM_OE_N_from_the_sram => SRAM_OE_N,
         SRAM_UB_N_from_the_sram => SRAM_UB_N,
         SRAM_WE_N_from_the_sram => SRAM_WE_N
);

end rtl;
```

DrawFacesModule.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity DrawFacesModule is
```

```
	port(
		clk : in std_logic;
		start : in std_logic;
		done : out std_logic;

		data		: out STD_LOGIC_VECTOR (35 DOWNTO 0);
		rdaddress		: out STD_LOGIC_VECTOR (9 DOWNTO 0);
		wren		: out STD_LOGIC  := '0';
		q		: in STD_LOGIC_VECTOR (35 DOWNTO 0);

		frame_address	: out STD_LOGIC_VECTOR (16 DOWNTO 0);
		frame_data		: out STD_LOGIC_VECTOR (1 DOWNTO 0);
		frame_wren		: out STD_LOGIC;

		backface_on : in std_logic;

		z_section : in std_logic_vector(5 downto 0);

		rotcontrol_z : in std_logic_vector(1 downto 0)

	);
end DrawFacesModule;


architecture rtl of DrawFacesModule is
	signal done_internal : std_logic := '0';
	type states is (IDLE_STATE, GETFACE_STATE, GETFACE2_STATE, GETFACE3_STATE,
GETFACE4_STATE, WAIT1_STATE, WAIT2_STATE, WAIT3_STATE, WAIT4_STATE, WAIT5_STATE,
WAIT6_STATE, BACKFACE_STATE, DRAW_STATE, DRAW2_STATE, DRAW3_STATE, DRAW4_STATE);
	signal state : states := IDLE_STATE;
	signal count, index, x_index, y_index, x_draw, y_draw, x_offset, y_offset,
row_count, vertex_count : integer := 0;
	signal vertex, vertex1, vertex2, vertex3, vertex4 : STD_LOGIC_VECTOR (35 DOWNTO
0);
	signal rotated_x, rotated_y, rotated_z, rotated_x1, rotated_y1, rotated_x2,
rotated_y2, rotated_x3, rotated_y3, rotated_x4, rotated_y4 : signed(10 downto 0);
	signal x_signed, y_signed : signed(35 downto 0);
   signal x0, y0, x1, y1 : signed(10 downto 0);
	signal line_start, line_done, line_reset, line_plot : std_logic := '0';
	signal x_p, y_p : signed(10 downto 0);
	signal scanline_data : std_logic_vector(1 downto 0);

	signal cos_value, sin_value : signed(19 downto 0);
	signal cos_x : signed(19 downto 0) := "00100000000000000000";
	signal sin_x : signed(19 downto 0) := "00110111011011001111";

	signal angle : integer := 0;

	signal sin_address, cos_address : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

```
	signal sin_q, cos_q : STD_LOGIC_VECTOR (19 DOWNTO 0);

	signal ux, uy, uz, vx, vy, vz : signed(10 downto 0);
	signal normal_z : signed(21 downto 0);
	signal face_angle : signed(32 downto 0);

	begin
		lines_inst : entity work.lines
		 port map (
			clk => clk,
			x0 => x0,
			y0 => y0,
			x1 => x1,
			y1 => y1,
			x_p => x_p,
			y_p => y_p,
			start => line_start,
			done => line_done,
			rst => line_reset,
			plot => line_plot

		 );


		process(clk)
		begin
			if rising_edge(clk) then
				case state is
				when IDLE_STATE =>
					done <= '0';
					wren <= '0';
					vertex1 <= q;
					count <= 0;

					if angle < 180 then
						sin_address <=
std_logic_vector(to_unsigned(angle, 8));
						cos_address <=
std_logic_vector(to_unsigned(angle, 8));
						cos_value <= signed(cos_q);
						sin_value <= signed(sin_q);
					else
						sin_address <=
std_logic_vector(to_unsigned((angle - 180), 8));
						cos_address <=
std_logic_vector(to_unsigned((angle - 180), 8));
						cos_value <= not signed(cos_q);
						sin_value <= not signed(sin_q);
					end if;
```

```
                        if angle >= 0 and angle < 90 then
                                index <= 0;
                                x_index <= 0;
                                y_index <= 0;
                                x_draw <= -96;
                                y_draw <= -96;
                        elsif angle >= 90 and angle < 180 then
                                index <= 960;
                                x_index <= 0;
                                y_index <= 30;
                                x_draw <= -96;
                                y_draw <= 84;
                        elsif angle >= 180 and angle < 270 then
                                index <= 990;
                                x_index <= 30;
                                y_index <= 30;
                                x_draw <= 84;
                                y_draw <= 84;
                        elsif angle >= 270 and angle < 360 then
                                index <= 30;
                                x_index <= 30;
                                y_index <= 0;
                                x_draw <= 84;
                                y_draw <= -96;
                        end if;
                        if start = '1' then
                                state <= GETFACE_STATE;
                        else
                                state <= IDLE_STATE;
                        end if;
                when GETFACE_STATE =>
                        if count < 961 then
                                vertex1 <= q;
                                state <= WAIT1_STATE;
                        else
                                if rotcontrol_z = "10" then
                                        if angle < 360 then
                                                angle <= angle + 1;
                                        else
                                                angle <= 0;
                                        end if;
                                elsif rotcontrol_z = "01" then
                                        if angle > 0 then
                                                angle <= angle - 1;
                                        else
                                                angle <= 360;
                                        end if;
                                else
                                        angle <= angle;
```

```
                end if;
                state <= IDLE_STATE;
                count <= 0;
                done <= '1';
        end if;
when WAIT1_STATE =>
        state <= GETFACE2_STATE;
        rotated_x1 <= rotated_x;
        rotated_y1 <= rotated_y;
when GETFACE2_STATE =>
        vertex2 <= q;
        state <= WAIT2_STATE;
when WAIT2_STATE =>
        state <= GETFACE3_STATE;
        rotated_x2 <= rotated_x;
        rotated_y2 <= rotated_y;
when GETFACE3_STATE =>
        vertex3 <= q;
        state <= WAIT3_STATE;
when WAIT3_STATE =>
        state <= GETFACE4_STATE;
        rotated_x3 <= rotated_x;
        rotated_y3 <= rotated_y;
when GETFACE4_STATE =>
        vertex4 <= q;
        state <= WAIT4_STATE;
when WAIT4_STATE =>
        rotated_x4 <= rotated_x;
        rotated_y4 <= rotated_y;
        state <= WAIT5_STATE;
when WAIT5_STATE =>
        if angle >= 0 and angle < 90 then
                if row_count = 30 then
                        row_count <= 0;
                        index <= index - 959;
                        x_index <= x_index + 1;
                        y_index <= 0;
                        x_draw <= x_draw + 6;
                        y_draw <= -96;
                else
                        row_count <= row_count + 1;
                        index <= index + 32;
                        y_index <= y_index + 1;
                        y_draw <= y_draw + 6;
                end if;
        elsif angle >= 90 and angle < 180 then
                if row_count = 30 then
                        row_count <= 0;
                        index <= index - 62;
```

```
                                        x_index <= 0;
                                        y_index <= y_index - 1;
                                        x_draw <= -96;
                                        y_draw <= y_draw - 6;
                                  else
                                        row_count <= row_count + 1;
                                        index <= index + 1;
                                        x_index <= x_index + 1;
                                        x_draw <= x_draw + 6;
                                  end if;
                            elsif angle >= 180 and angle < 270 then
                                  if row_count = 30 then
                                        row_count <= 0;
                                        index <= index + 959;
                                        x_index <= x_index - 1;
                                        y_index <= 30;
                                        x_draw <= x_draw - 6;
                                        y_draw <= 84;
                                  else
                                        row_count <= row_count + 1;
                                        index <= index - 32;
                                        y_index <= y_index - 1;
                                        y_draw <= y_draw - 6;
                                  end if;
                            elsif angle >= 270 and angle < 360 then
                                  if row_count = 30 then
                                        row_count <= 0;
                                        index <= index + 62;
                                        x_index <= 30;
                                        y_index <= y_index + 1;
                                        x_draw <= 84;
                                        y_draw <= y_draw + 6;
                                  else
                                        row_count <= row_count + 1;
                                        index <= index - 1;
                                        x_index <= x_index - 1;
                                        x_draw <= x_draw - 6;
                                  end if;
                            end if;
                            count <= count + 1;
                            state <= WAIT6_STATE;
                      when WAIT6_STATE =>
                            state <= BACKFACE_STATE;
                      when BACKFACE_STATE =>
                            if normal_z > to_signed(0, 22) or (normal_z <=
to_signed(0, 22) and backface_on = '0') then
                                  if count < 160 and z_section(5) = '1' then
                                        frame_data <= "01";
                                        state <= DRAW_STATE;
```

```
                                        elsif count >= 160 and count < 320 and
z_section(4) = '1' then
                                              frame_data <= "10";
                                              state <= DRAW_STATE;
                                        elsif count >= 320 and count < 480 and
z_section(3) = '1' then
                                              frame_data <= "11";
                                              state <= DRAW_STATE;
                                        elsif count >= 480 and count < 640 and
z_section(2) = '1' then
                                              frame_data <= "01";
                                              state <= DRAW_STATE;
                                        elsif count >= 640 and count < 800 and
z_section(1) = '1' then
                                              frame_data <= "10";
                                              state <= DRAW_STATE;
                                        elsif count >= 800 and count < 961 and
z_section(0) = '1' then
                                              frame_data <= "11";
                                              state <= DRAW_STATE;
                                        else
                                              frame_data <= "01";
                                              state <= GETFACE_STATE;
                                        end if;
                                  else
                                        frame_data <= "01";
                                        state <= GETFACE_STATE;
                                  end if;
                            when DRAW_STATE =>
                                  if line_done = '0' then
                                        x0 <= rotated_x1 + to_signed(160, 11);
                                        y0 <= rotated_y1 + to_signed(180, 11);
                                        x1 <= rotated_x2 + to_signed(160, 11);
                                        y1 <= rotated_y2 + to_signed(180, 11);
                                        line_reset <= '0';
                                        line_start <= '1';
                                  else
                                        line_start <= '0';
                                        line_reset <= '1';
                                        state <= DRAW2_STATE;
                                  end if;
                            when DRAW2_STATE =>
                                  if line_done = '0' then
                                        x0 <= rotated_x1 + to_signed(160, 11);
                                        y0 <= rotated_y1 + to_signed(180, 11);
                                        x1 <= rotated_x3 + to_signed(160, 11);
                                        y1 <= rotated_y3 + to_signed(180, 11);
                                        line_reset <= '0';
                                        line_start <= '1';
```

```
                                else
                                        line_start <= '0';
                                        line_reset <= '1';
                                        state <= DRAW3_STATE;
                                end if;
                        when DRAW3_STATE =>
                                if line_done = '0' then
                                        x0 <= rotated_x2 + to_signed(160, 11);
                                        y0 <= rotated_y2 + to_signed(180, 11);
                                        x1 <= rotated_x4 + to_signed(160, 11);
                                        y1 <= rotated_y4 + to_signed(180, 11);
                                        line_reset <= '0';
                                        line_start <= '1';
                                else
                                        line_start <= '0';
                                        line_reset <= '1';
                                        state <= DRAW4_STATE;
                                end if;
                        when DRAW4_STATE =>
                                if line_done = '0' then
                                        x0 <= rotated_x3 + to_signed(160, 11);
                    y0 <= rotated_y3 + to_signed(180, 11);
                                        x1 <= rotated_x4 + to_signed(160, 11);
                                        y1 <= rotated_y4 + to_signed(180, 11);
                                        line_reset <= '0';
                                        line_start <= '1';
                                else
                                        line_start <= '0';
                                        line_reset <= '1';
                                        state <= GETFACE_STATE;
                                end if;
                        end case;
                  end if;
            end process;

            frame_address <= std_logic_vector(y_p * 320 + x_p)(16 downto 0);
            frame_wren <= line_plot;
            rdaddress <= std_logic_vector(to_unsigned(index + vertex_count, 10));
            vertex_count <= 1 when state = GETFACE_STATE or state = WAIT1_STATE else
                                                32 when state = GETFACE2_STATE or
state = WAIT2_STATE else
                                                33 when state = GETFACE3_STATE or
state = WAIT3_STATE else 0;
            x_offset <= 6 when state = GETFACE2_STATE or state = WAIT2_STATE or state
= GETFACE4_STATE or state = WAIT4_STATE else 0;
            y_offset <= 6 when state = GETFACE3_STATE or state = WAIT3_STATE or state
= GETFACE4_STATE or state = WAIT4_STATE else 0;

            rotated_x <= signed(x_signed * cos_value + y_signed * not sin_value)(46
```

```
downto 36);
              rotated_y <= signed(y_signed * cos_x * cos_value + signed(vertex(35
downto 0)) * not sin_x * to_signed(262144, 20) + x_signed * cos_x * sin_value)(64
downto 54);


              ux <= rotated_x2 - rotated_x1;
              uy <= rotated_y2 - rotated_y1;
              vx <= rotated_x3 - rotated_x1;
              vy <= rotated_y3 - rotated_y1;
              normal_z <= ux*vy - uy*vx;

              vertex <= vertex1 when state = WAIT1_STATE else
                                    vertex2 when state = WAIT2_STATE else
                                    vertex3 when state = WAIT3_STATE else
                                    vertex4 when state = WAIT4_STATE;

              x_signed <= to_signed((x_draw + x_offset), 18) & "00000000000000000";
              y_signed <= to_signed((y_draw + y_offset), 18) & "00000000000000000";

              sin_memory : entity work.SinLookup port map(
                     address => sin_address,
                     clock => clk,
                     q => sin_q
              );

              cos_memory : entity work.CosLookup port map(
                     address => cos_address,
                     clock => clk,
                     q => cos_q
              );

end rtl;
```

lines.vhd (modified from example)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lines is
  port (
    clk : in std_logic;
    rst : in std_logic;
    x0, y0, x1, y1 : in signed(10 downto 0);
    x_p, y_p : out signed(10 downto 0);
    start : in std_logic;
    done, plot : out std_logic
```

```
  );

end lines;

architecture datapath of lines is
  signal x1x0, y1y0, dx, dy : signed(10 downto 0);
  signal down, right, e2_lt_dx, e2_gt_dy :std_logic;
  signal in_loop, break : std_logic;

  signal err, err1, err2, e2, err_next : signed(11 downto 0);
  signal x, y, x_next, y_next, xa, xb, ya, yb : signed(10 downto 0);

  type states is (IDLE_STATE, RUNNING_STATE, DONE_STATE);

  signal state : states;
begin

  process (clk)
  begin
    if rising_edge(clk) then
      err <= err_next;
      x <= x_next;
      y <= y_next;
    end if;
  end process;

  fsm : process (clk)
  begin
    if rising_edge(clk) then
      if rst = '1' then
        state <= IDLE_STATE;
      else
        case state is
          when IDLE_STATE =>
            if start = '1' then
              state <= RUNNING_STATE;
            end if;
          when RUNNING_STATE =>
            if break = '1' then
              state <= DONE_STATE;
            end if;
          when DONE_STATE =>
            if start = '1' then
              state <= RUNNING_STATE;
            else
              state <= IDLE_STATE;
            end if;
        end case;
      end if;
```

```
    end if;
  end process;

  in_loop <= '1' when state = RUNNING_STATE else '0';
  plot <= in_loop;
  done <= '1' when state = DONE_STATE else '0';

  x1x0 <= x1 - x0;
  y1y0 <= y1 - y0;

  right <= not x1x0(10);
  down <= not y1y0(10);

  dx <= x1x0 when right = '1' else -x1x0;
  dy <= -y1y0 when down = '1' else y1y0;

  err_next <= ("0" & dx) + ("0" & dy) when in_loop = '0' else err2;

  err2 <= err1 + dx when e2_lt_dx = '1' else err1;

  err1 <= err + dy when e2_gt_dy = '1' else err;

  x_next <= x0 when in_loop = '0' else xb;
  y_next <= y0 when in_loop = '0' else yb;

  xb <= xa when e2_gt_dy = '1' else x;
  yb <= ya when e2_lt_dx = '1' else y;

  xa <= x + 1 when right = '1' else x - 1;
  ya <= y + 1 when down = '1' else y - 1;

  e2 <= err(10 downto 0) & "0";

  e2_gt_dy <= '1' when e2 > dy else '0';
  e2_lt_dx <= '1' when e2 < dx else '0';

  break <= '1' when x = x1 and y = y1 else '0';

  x_p <= x;
  y_p <= y;

end datapath;
```

ImageToVga.vhd

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;

entity ImageToVGA is
	port(
		signal clk : in std_logic;
		signal reset : in std_logic;
		signal FRAME_DONE : out std_logic;
		signal VALUE_AT_CUR : in std_logic_vector(1 downto 0);

		signal XPOS : out std_logic_vector(9 downto 0);
		signal YPOS : out std_logic_vector(9 downto 0);
		signal position : out std_logic_vector(18 downto 0);
		signal color_options : in std_logic_vector(2 downto 0);

		VGA_CLK,
		VGA_HS,
		VGA_VS,
		VGA_BLANK,
		VGA_SYNC : out std_logic;
		VGA_R,
		VGA_G,
		VGA_B : out unsigned(9 downto 0)
	);
end ImageToVGA;

architecture rtl of ImageToVGA is
  constant HTOTAL       : integer := 800;
  constant HSYNC        : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE      : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant HALFHLENGTH : integer := 10;
  constant HALFVLENGTH : integer := 10;

  constant VTOTAL       : integer := 525;
  constant VSYNC        : integer := 2;
  constant VBACK_PORCH  : integer := 33;
  constant VACTIVE      : integer := 480;
  constant VFRONT_PORCH : integer := 10;

  signal Hcount : unsigned(9 downto 0);
  signal Vcount : unsigned(9 downto 0);

  signal count : integer := 0;

  signal EndOfLine, EndOfField : std_logic;
  signal vga_hblank, vga_hsync,
    vga_vblank, vga_vsync : std_logic;
```

```
  signal gradient : integer := 0;

begin
       HCounter : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        Hcount <= (others => '0');
      elsif EndOfLine = '1' then
        Hcount <= (others => '0');
      else
        Hcount <= Hcount + 1;
      end if;

              position <= std_logic_vector(to_unsigned(count, 19));
    end if;
  end process HCounter;


  EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

  VCounter: process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        Vcount <= (others => '0');
      elsif EndOfLine = '1' then
        if EndOfField = '1' then
          Vcount <= (others => '0');
                    gradient <= 0;
        else
          Vcount <= Vcount + 1;
                    gradient <= gradient + 1;
        end if;
      end if;
    end if;
  end process VCounter;


  EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

  HSyncGen : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' or EndOfLine = '1' then
        vga_hsync <= '1';
      elsif Hcount = HSYNC - 1 then
        vga_hsync <= '0';
```

```
    end if;
  end if;
end process HSyncGen;

HBlankGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_hblank <= '1';
    elsif Hcount = HSYNC + HBACK_PORCH then
      vga_hblank <= '0';
    elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
      vga_hblank <= '1';
    end if;
  end if;
end process HBlankGen;

VSyncGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_vsync <= '1';
    elsif EndOfLine ='1' then
      if EndOfField = '1' then
        vga_vsync <= '1';
      elsif Vcount = VSYNC - 1 then
        vga_vsync <= '0';
      end if;
    end if;
  end if;
end process VSyncGen;

VBlankGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_vblank <= '1';
    elsif EndOfLine = '1' then
      if Vcount = VSYNC + VBACK_PORCH - 1 then
        vga_vblank <= '0';
      elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
        vga_vblank <= '1';
      end if;
    end if;
  end if;
end process VBlankGen;

RasterGen : process(clk)
begin
```

```
       if rising_edge(clk) then
              if((Hcount >= HSYNC + HBACK_PORCH) and (Hcount < HSYNC + HBACK_PORCH +
HACTIVE)) then
                     XPOS <= std_logic_vector(Hcount - HSYNC - HBACK_PORCH);
              end if;
              if((Vcount >= VSYNC + VBACK_PORCH - 1) and (Vcount < VSYNC + VBACK_PORCH
+ VACTIVE - 1) and (Hcount >= HSYNC + HBACK_PORCH) and (Hcount < HSYNC + HBACK_PORCH +
HACTIVE)) then
                     YPOS <= std_logic_vector(Vcount - VSYNC - VBACK_PORCH + 1);
                     if((Vcount - VSYNC - VBACK_PORCH + 1) mod 2 = 0 and (Hcount -
HSYNC - HBACK_PORCH) mod 2 = 0) then
                            count <= count + 1;
                     end if;
              end if;
              if EndOfField = '1' then
                     count <= 0;
              end if;
              FRAME_DONE <= EndOfField;
       end if;
  end process RasterGen;



  VideoOut: process (clk, reset)
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif clk'event and clk = '1' then
              if color_options = "000" then
                     if VALUE_AT_CUR = "00" then
                       VGA_R <= to_unsigned(gradient, 10);
                       VGA_G <= "0010000000";
                       VGA_B <= "0000000000";
                     elsif VALUE_AT_CUR = "10" then
                       VGA_R <= "1111111111";
                       VGA_G <= "0000000000";
                       VGA_B <= "1111111111";
                     elsif VALUE_AT_CUR = "01" then
                       VGA_R <= "0000000000";
                       VGA_G <= "1111111111";
                       VGA_B <= "1111111111";
                     elsif VALUE_AT_CUR = "11" then
                       VGA_R <= "1111111111";
                       VGA_G <= "1111111111";
                       VGA_B <= "1111111111";
                     elsif vga_hblank = '0' and vga_vblank ='0' then
                       VGA_R <= "0000000000";
```

```
            VGA_G <= "0000000000";
            VGA_B <= "0000000000";
          else
            VGA_R <= "0000000000";
            VGA_G <= "0000000000";
            VGA_B <= "0000000000";
          end if;
        elsif color_options = "001" then
          if VALUE_AT_CUR = "00" then
            VGA_R <= to_unsigned(gradient, 10);
            VGA_G <= "0010000000";
            VGA_B <= "0000000000";
          elsif VALUE_AT_CUR = "10" then
            VGA_R <= "0000000000";
            VGA_G <= "1111111111";
            VGA_B <= to_unsigned(gradient + 200, 10);
          elsif VALUE_AT_CUR = "01" then
            VGA_R <= "0000000000";
            VGA_G <= "1111111111";
            VGA_B <= to_unsigned(gradient + 200, 10);
          elsif VALUE_AT_CUR = "11" then
            VGA_R <= "1111111111";
            VGA_G <= "1111111111";
            VGA_B <= "1111111111";
          elsif vga_hblank = '0' and vga_vblank ='0' then
            VGA_R <= "0000000000";
            VGA_G <= "0000000000";
            VGA_B <= "0000000000";
          else
            VGA_R <= "0000000000";
            VGA_G <= "0000000000";
            VGA_B <= "0000000000";
          end if;
        elsif color_options = "010" then
          if VALUE_AT_CUR = "00" then
            VGA_R <= to_unsigned(gradient, 10);
            VGA_G <= "0010000000";
            VGA_B <= "0000000000";
          elsif VALUE_AT_CUR = "10" then
            VGA_R <= "0000000000";
            VGA_G <= "1111111111";
            VGA_B <= "1111111111";
          elsif VALUE_AT_CUR = "01" then
            VGA_R <= "0000000000";
            VGA_G <= "1111111111";
            VGA_B <= "1111111111";
          elsif VALUE_AT_CUR = "11" then
            VGA_R <= "0000000000";
            VGA_G <= "1111111111";
```

```
                VGA_B <= "1111111111";
            elsif vga_hblank = '0' and vga_vblank ='0' then
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
            else
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
            end if;
        elsif color_options = "100" then
            if VALUE_AT_CUR = "00" then
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
            elsif VALUE_AT_CUR = "10" then
                VGA_R <= "1111111111";
                VGA_G <= "1111111111";
                VGA_B <= "1111111111";
            elsif VALUE_AT_CUR = "01" then
                VGA_R <= "1111111111";
                VGA_G <= "1111111111";
                VGA_B <= "1111111111";
            elsif VALUE_AT_CUR = "11" then
                VGA_R <= "1111111111";
                VGA_G <= "1111111111";
                VGA_B <= "1111111111";
            elsif vga_hblank = '0' and vga_vblank ='0' then
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
            else
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
            end if;
        else
            if VALUE_AT_CUR = "00" then
                VGA_R <= to_unsigned(gradient, 10);
                VGA_G <= "0010000000";
                VGA_B <= "0000000000";
            elsif VALUE_AT_CUR = "10" then
                VGA_R <= "0000000000";
                VGA_G <= "1111111111";
                VGA_B <= "1111111111";
            elsif VALUE_AT_CUR = "01" then
                VGA_R <= "0000000000";
                VGA_G <= "1111111111";
                VGA_B <= "1111111111";
```

```
                    elsif VALUE_AT_CUR = "11" then
                      VGA_R <= "1111111111";
                      VGA_G <= "1111111111";
                      VGA_B <= "1111111111";
                    elsif vga_hblank = '0' and vga_vblank ='0' then
                      VGA_R <= "0000000000";
                      VGA_G <= "0000000000";
                      VGA_B <= "0000000000";
                    else
                      VGA_R <= "0000000000";
                      VGA_G <= "0000000000";
                      VGA_B <= "0000000000";
                    end if;
              end if;
    end if;
  end process VideoOut;

  VGA_CLK <= clk;
  VGA_HS <= not vga_hsync;
  VGA_VS <= not vga_vsync;
  VGA_SYNC <= '0';
  VGA_BLANK <= not (vga_hsync or vga_vsync);

end rtl;
```

de2_cpu_to_vga.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_cpu_to_vga is

  port (
    clk        : in  std_logic;
    reset_n    : in  std_logic;
    read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  std_logic_vector(5 downto 0);
    readdata   : out std_logic_vector(31 downto 0);
    writedata  : in  std_logic_vector(31 downto 0);

        gen_start : in std_logic;
        gen_done : out std_logic := '0';
        hmap_address : out std_logic_vector(9 downto 0);
        data : out std_logic_vector(35 downto 0);
```

```
       wren : out std_logic
  );

end de2_cpu_to_vga;

architecture rtl of de2_cpu_to_vga is
  type ram_type is array(31 downto 0) of
            std_logic_vector(31 downto 0);
  signal RAM : ram_type;
  signal ram_address, display_address : unsigned(4 downto 0);
  signal xpos : integer := 0;
  signal ypos : integer := 0;


begin
  ram_address <= unsigned(address(4 downto 0));

  process(clk)
  begin
    if rising_edge(clk) then
      if chipselect = '1' then
                    if read = '1' then
                                  readdata <= RAM(to_integer(ram_address));
                    elsif write = '1' then
                           RAM(to_integer(ram_address)) <= writedata;
                end if;
      end if;
             RAM(3) <= "00000000000000000000000000000000" & gen_start;
        end if;
  end process;

       xpos <= to_integer(signed(RAM(0)));
       ypos <= to_integer(signed(RAM(1)));
       hmap_address <= std_logic_vector(to_unsigned(ypos * 32 + xpos, 10));
       data <= "0000" & RAM(2);
       wren <= '1' when write = '1' and to_integer(ram_address) = 2 else '0';
       gen_done <= RAM(4)(0);
end rtl;
```

7.2 Testbenches

lines_tb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;

entity lines_tb is
end lines_tb;

architecture tb of lines_tb is
	signal clk : std_logic := '0';
	signal rst : std_logic;
	signal x0, y0, x1, y1 : signed(17 downto 0);
	signal x_p, y_p : signed(17 downto 0);
	signal start : std_logic;
	signal step : std_logic;
	signal done, plot : std_logic;

	begin
	clk <= not clk after 10 ns;

	process
	begin
		wait for 20 ns;
		start <= '1';
		rst <= '1';
		x0 <= to_signed(0, 18);
		y0 <= to_signed(0, 18);
		x1 <= to_signed(40, 18);
		y1 <= to_signed(20, 18);
		step <= '1';
		wait for 20 ns;
		rst <= '0';
		wait;
	end process;

	test_lines : entity work.lines
		port map (
			clk => clk,
			rst => rst,
			x0 => x0,
			y0 => y0,
			x1 => x1,
			y1 => y1,
			x_p => x_p,
			y_p => y_p,
			done => done,
			plot => plot,
			start => start
			--step => step
		);
end tb;
```

image_to_vga_tb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.mytypes.all;

entity image_to_vga_tb is
end image_to_vga_tb;

architecture tb of image_to_vga_tb is
	signal clk : std_logic := '0';
	signal reset : std_logic := '1';
	signal FRAME_DONE : std_logic;
	signal VALUE_AT_CUR : std_logic := '1';

		--signal XPOS : out std_logic_vector(9 downto 0);
		--signal YPOS : out std_logic_vector(9 downto 0);
	signal position : std_logic_vector(18 downto 0);

	signal
		VGA_CLK,                                        -- Clock
		VGA_HS,                                         -- H_SYNC
		VGA_VS,                                         -- V_SYNC
		VGA_BLANK,                                      -- BLANK
		VGA_SYNC : std_logic;                        -- SYNC
	signal
		VGA_R,                                          -- Red[9:0]
		VGA_G,                                          -- Green[9:0]
		VGA_B : unsigned(9 downto 0);                -- Blue[9:0]


	 signal hmap_data : std_logic_vector(53 downto 0);
	 signal hmap_rdaddress : std_logic_vector(9 downto 0);
	 signal hmap_wraddress : std_logic_vector(9 downto 0);
	 signal hmap_wren : std_logic;
	 signal hmap_q : std_logic_vector(53 downto 0);
	 signal hmap_start : std_logic := '1';


  signal vbuffer_row_write : std_logic_vector(0 downto 0);
  signal vbuffer_wren : std_logic := '0';
  signal vbuffer_write_addr : std_logic_vector(18 downto 0);

  signal start_raster : std_logic := '1';
  signal done_raster : std_logic := '0';

	signal point_out : std_logic;
```

```
	begin
		clk <= not clk after 10 ns;
		process
		begin
			wait for 20 ns;
			reset <= '0';
			wait;
		end process;

		test_image_to_vga : entity work.ImageToVGA
		port map (
			clk => clk,
			reset => reset,
			FRAME_DONE => FRAME_DONE,
			VALUE_AT_CUR => VALUE_AT_CUR,
			position => position,
			VGA_CLK => VGA_CLK,
			VGA_HS => VGA_HS,
			VGA_VS => VGA_VS,
			VGA_BLANK => VGA_BLANK,
			VGA_SYNC => VGA_SYNC,
			VGA_R => VGA_R,
			VGA_G => VGA_G,
			VGA_B => VGA_B,
			point_out => point_out
		);

		image_from_heightmap : entity work.ImageFromHeightMap port map(
			clk => clk,
			--XPOS => xpos,
			--YPOS => ypos,
			position => position,
			curr_value => VALUE_AT_CUR,
			data_in => vbuffer_row_write,
			wren => vbuffer_wren,
			wraddress => vbuffer_write_addr,
			start => hmap_start
		);

		test_heightmap_memory : entity work.HeightMapMemory port map(
			clock => clk,
			data => hmap_data,
			rdaddress => hmap_rdaddress,
			wraddress => hmap_wraddress,
			wren => hmap_wren,
			q => hmap_q
		);

		rastermodule : entity work.RasterizeModule port map(
```

```
                    clk => clk,
                    start => start_raster,
                    done => done_raster,

                    rdaddress => hmap_rdaddress,
                    q => hmap_q,

                    vbuffer_data_out => vbuffer_row_write,
                    vbuffer_wren => vbuffer_wren,
                    vbuffer_wraddress => vbuffer_write_addr
            );
end tb;
```

raster_tb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity raster_tb is
end raster_tb;

architecture tb of raster_tb is
            signal x_signed, y_signed : signed(35 downto 0);
            signal x_index, y_index : integer := 0;
            signal offset : integer := -16;

      begin
            --clk <= not clk after 10 ns;
            process
            begin
                  wait for 100 ns;
                  x_index <= 0;
                  y_index <= 0;
                  wait for 20 ns;
                  x_index <= 4;
                  y_index <= 1;
                  wait for 20 ns;
```

```
                    x_index <= 16;
                    y_index <= 16;
                    wait for 20 ns;
                    x_index <= 20;
                    y_index <= 20;
                    wait for 20 ns;
                    x_index <= 15;
                    y_index <= 15;
                    wait for 20 ns;
                    x_index <= 30;
                    y_index <= 30;

                    wait;
            end process;

            x_signed <= to_signed((x_index + offset) * 6, 18) & "000000000000000000";
            y_signed <= to_signed((y_index + offset) * 6, 18) & "000000000000000000";
end tb;
```

7.3 NIOS II C Code

hmap_gen.c

```
#include <io.h>
#include <system.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define IOWR_VGA_DATA(base, offset, data) \
        IOWR_32DIRECT(base, (offset) * 4, data)
#define IORD_VGA_DATA(base, offset) \
        IORD_32DIRECT(base, (offset) * 4)

void iterativePass(int iterLength, double** grid){
       printf("start\n");
       int i, j;
       int max = 8;
```

```
	for(i = 0; i < 33 - iterLength; i+=iterLength){
		for( j = 0; j < 33 - iterLength; j+=iterLength){
			int centerX = iterLength/2 + i;
			int centerY = iterLength/2 + j;
			grid[centerX][centerY] = (grid[i][j] + grid[i+iterLength][j] +
grid[i][j+iterLength] + grid[i+iterLength][j + iterLength])/4 + rand() % (max);
		}
	}
	printf("part1 done\n");
	for(i = 0; i < 33 - iterLength; i+=iterLength){
		for(j = 0; j < 33 - iterLength; j+=iterLength){
			grid[i + iterLength/2][j] = (grid[i][j] + grid[i +
iterLength][j])/2 + rand() % (max);
			grid[i + iterLength][j + iterLength/2] = (grid[i + iterLength][j]
+ grid[i + iterLength][j + iterLength])/2 + rand() % (max);
			grid[i + iterLength/2][j + iterLength] = (grid[i][j + iterLength]
+ grid[i + iterLength][j + iterLength])/2 + rand() % (max);
			grid[i][j + iterLength/2] = (grid[i][j] + grid[i][j +
iterLength])/2 + rand() % (max);
		}
	}
	printf("part2 done\n");
}

void startIterative(double **grid){
	//int grid[33][33];
	int i;
	int gridEnd = 32;
	int max = 128;
	grid[0][0] = rand() % max;
	grid[0][gridEnd] = rand() % max;
	grid[gridEnd][0] = rand() % max;
	grid[gridEnd][gridEnd] = rand() % max;

	int loops = 5;
	int iterLength = gridEnd;
	for(i = 0; i < loops && iterLength > 0; i++){
		iterativePass(iterLength, grid);
		iterLength /= 2;
		printf("pass: %d\n", i);
	}
	iterLength *= 2;
}

int main(){
	srand(time(NULL));
	double** grid = (double**)malloc(33 * sizeof(int*));
	int i,j;
	for(i = 0; i < 33; i++){
```

```
		grid[i] = (double*)malloc(33*sizeof(int));
	}
	while(1){
		if(IORD_VGA_DATA(0x101000, 3) == 1){
			startIterative(grid);
			for(i = 0; i < 32; i++){
				for(j = 0; j < 32; j++){
					IOWR_VGA_DATA(DE2_CPU_TO_VGA_0_BASE, 0, i);
					IOWR_VGA_DATA(DE2_CPU_TO_VGA_0_BASE, 1, j);
					IOWR_VGA_DATA(DE2_CPU_TO_VGA_0_BASE, 2,
(int)grid[i][j] * 262144);

				}
			}
			IOWR_VGA_DATA(DE2_CPU_TO_VGA_0_BASE, 4, 1);
			IOWR_VGA_DATA(DE2_CPU_TO_VGA_0_BASE, 4, 0);
			printf("%d\n", rand() % 32);
		}

	}
	free(grid);

	return 0;
}
```

7.4 C Code to Create .mif Memory Initialization Files

lookup_generator.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

void get_binary(long decimal, char* buffer);

int main(int argc, char* argv[]){
	FILE *sin_file, *cos_file;

	sin_file = fopen("sin_lookup.mif", "wb+");
	cos_file = fopen("cos_lookup.mif", "wb+");

	if(sin_file != NULL){
		fputs("WIDTH=36;\n", sin_file);
		fputs("DEPTH=360;\n", sin_file);
		fputs("\nADDRESS_RADIX=UNS;\n", sin_file);
```

```
		fputs("DATA_RADIX=BIN;\n", sin_file);
		fputs("\nCONTENT BEGIN\n", sin_file);
		int i;
		double current_angle;
		double current_sin;
		long truncated_sin;
		char current_binary[37];
		for(i = 0; i < 360; i++){
			current_angle = i * M_PI/180;
			current_sin = sin(current_angle / 4) * 262144;
			truncated_sin = (long)current_sin;
			get_binary(truncated_sin, current_binary);
			fprintf(sin_file, "%d : %s;\n", i, current_binary[16]);
			printf("%d %f %s\n", truncated_sin, sin(current_angle),
current_binary);

		}
		fputs("END;", sin_file);
	}

	if(cos_file != NULL){
		fputs("WIDTH=36;\n", cos_file);
		fputs("DEPTH=360;\n", cos_file);
		fputs("\nADDRESS_RADIX=UNS;\n", cos_file);
		fputs("DATA_RADIX=BIN;\n", cos_file);
		fputs("\nCONTENT BEGIN\n", cos_file);
		int i;
		double current_angle, current_cos;
		long truncated_cos;
		char current_binary[37];
		for(i = 0; i < 360; i++){
			current_angle = i * M_PI/180;
			current_cos = cos(current_angle / 4) * 262144;
			truncated_cos = (int)current_cos;
			get_binary(truncated_cos, current_binary);
			fprintf(cos_file, "%d : %s;\n", i, current_binary[16]);
			printf("%d %f %s\n", truncated_cos, cos(current_angle),
current_binary);
		}
		fputs("END;", cos_file);
	}
}

void get_binary(long decimal, char* buffer){
	int i;
	buffer[36] = '\0';
	if(decimal < 0){
		buffer[0] = '1';
		long current_value = -powl(2, 35);
```

```
		for(i = 1; i < 36; i++){
			if(current_value + powl(2, 35 - i) <= decimal){
				current_value += powl(2, 35 - i);
				buffer[i] = '1';
			}
			else{
				buffer[i] = '0';
			}
		}
	}
	else{
		buffer[0] = '0';
		for(i = 1; i < 36; i++){
			if(decimal - powl(2, 35 - i) >= 0){
				decimal -= powl(2, 35 - i);
				buffer[i] = '1';
			}
			else{
				buffer[i] = '0';
			}
		}
	}
}
```

test_hmap.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

void get_binary(long decimal, char* buffer);
double** startIterative();
void iterativePass(int iterLength, double** grid);


int main(int argc, char* argv[]){
	double** grid = startIterative();
	int i, j;

	FILE *test_hmap;
	test_hmap = fopen("test_hmap.mif", "wb+");

	if(test_hmap != NULL){
		fputs("WIDTH=36;\n", test_hmap);
		fputs("DEPTH=1024;\n", test_hmap);
		fputs("\nADDRESS_RADIX=UNS;\n", test_hmap);
		fputs("DATA_RADIX=BIN;\n", test_hmap);
```

```
		fputs("\nCONTENT BEGIN\n", test_hmap);

		int truncated_height;
		char bin_z[37];
		char bin_x[37];
		char bin_y[37];
		long dist_interval = 6;
		int count = 0;
		for(j = 0; j < 32; j++){
			for(i = 0; i < 32; i++){
				truncated_height = (int)(grid[i][j] * 262144);
				printf("test\n");
				get_binary(262144 * (i - 16) * dist_interval, bin_x);
				get_binary(262144 * (j - 16) * dist_interval, bin_y);
				//get_binary(262144, bin_z);
				get_binary(truncated_height, bin_z);
				//fprintf(test_hmap, "%d : %s%s%s;\n", count, bin_x, bin_y,
bin_z);
				fprintf(test_hmap, "%d : %s;\n", count, bin_z);
				printf("%d, %d, %s\n", i, j, bin_z);
				count++;
			}
		}
		fputs("END;", test_hmap);
	}
}

void get_binary(long decimal, char* buffer){
	int i;
	buffer[36] = '\0';
	if(decimal < 0){
		buffer[0] = '1';
		long current_value = -powl(2, 35);
		for(i = 1; i < 36; i++){
			if(current_value + powl(2, 35 - i) <= decimal){
				current_value += powl(2, 35 - i);
				buffer[i] = '1';
			}
			else{
				buffer[i] = '0';
			}
		}
	}
	else{
		buffer[0] = '0';
		for(i = 1; i < 36; i++){
			if(decimal - powl(2, 35 - i) >= 0){
				decimal -= powl(2, 35 - i);
				buffer[i] = '1';
```

```
                        }
                        else{
                                buffer[i] = '0';
                        }
                }
        }
}

double** startIterative(){
        srand(time(NULL));
        //int grid[33][33];

        double** grid = (double**)malloc(34 * sizeof(int*));
        int i;
        for(i = 0; i < 34; i++){
                grid[i] = (double*)malloc(34*sizeof(int));
        }

        int gridEnd = 33;
        int max = 6;
        grid[0][0] = rand() % max;
        grid[0][gridEnd] = rand() % max;
        grid[gridEnd][0] = rand() % max;
        grid[gridEnd][gridEnd] = rand() % max;

        int loops = 5;
        int iterLength = gridEnd;
        for(i = 0; i < loops && iterLength > 0; i++){
                iterativePass(iterLength, grid);
                iterLength /= 2;
        }
        iterLength *= 2;
        return grid;
}

void iterativePass(int iterLength, double** grid){
        int i, j;
        int max = 33;
        for(i = 0; i < 34 - iterLength; i+=iterLength){
                for( j = 0; j < 34 - iterLength; j+=iterLength){
                        int centerX = iterLength/2 + i;
                        int centerY = iterLength/2 + j;
                        grid[centerX][centerY] = (grid[i][j] + grid[i+iterLength][j] +
grid[i][j+iterLength] + grid[i+iterLength][j + iterLength])/4 + rand() % (max/2);
                }
        }
        for(i = 0; i < 34 - iterLength; i+=iterLength){
                for(j = 0; j < 34 - iterLength; j+=iterLength){
                        grid[i + iterLength/2][j] = (grid[i][j] + grid[i +
```

```
iterLength][j])/2 + rand() % (max/2);
                    grid[i + iterLength][j + iterLength/2] = (grid[i + iterLength][j]
+ grid[i + iterLength][j + iterLength])/2 + rand() % (max/2);
                    grid[i + iterLength/2][j + iterLength] = (grid[i][j + iterLength]
+ grid[i + iterLength][j + iterLength])/2 + rand() % (max/2);
                    grid[i][j + iterLength/2] = (grid[i][j] + grid[i][j +
iterLength])/2 + rand() % (max/2);
             }
      }
}
```

7.5 Memory Initialization Files

sin_lookup.mif

```
WIDTH=20;
DEPTH=181;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;

CONTENT BEGIN
0 : 00000000000000000000;
1 : 00000001000111011111;
2 : 00000010001110111100;
3 : 00000011010110010111;
4 : 00000100011101101110;
5 : 00000101100100111111;
6 : 00000110101100001001;
7 : 00000111110011001011;
8 : 00001000111010000011;
9 : 00001010000000110000;
10 : 00001011000111010000;
11 : 00001100001101100011;
12 : 00001101010011100110;
13 : 00001110011001011001;
14 : 00001111011110111010;
15 : 00010000100100000111;
16 : 00010001101001000000;
17 : 00010010101101100011;
18 : 00010011110001101110;
19 : 00010100110101100001;
20 : 00010101111000111010;
```

```
21 : 00010110111011111000;
22 : 00010111111110011000;
23 : 00011001000000011011;
24 : 00011010000001111111;
25 : 00011011000011000010;
26 : 00011100000011100100;
27 : 00011101000011100010;
28 : 00011110000010111101;
29 : 00011111000001110001;
30 : 00011111111111111111;
31 : 00100000111101100110;
32 : 00100001111010100011;
33 : 00100010110110110101;
34 : 00100011110010011101;
35 : 00100100101101010111;
36 : 00100101100111100100;
37 : 00100110100001000010;
38 : 00100111011001101111;
39 : 00101000010001101100;
40 : 00101001001000110110;
41 : 00101001111111001101;
42 : 00101010110100110000;
43 : 00101011101001011101;
44 : 00101100011101010100;
45 : 00101101010000010011;
46 : 00101110000010011010;
47 : 00101110110011100111;
48 : 00101111100011111010;
49 : 00110000010011010010;
50 : 00110001000001101101;
51 : 00110001101111001100;
52 : 00110010011011101100;
53 : 00110011000111001101;
54 : 00110011110001101110;
55 : 00110100011011001111;
56 : 00110101000011101111;
57 : 00110101101011001100;
58 : 00110110010001100110;
59 : 00110110110110111101;
60 : 00110111011011001111;
61 : 00110111111110011100;
62 : 00111000100000100011;
63 : 00111001000001100100;
64 : 00111001100001011101;
65 : 00111010000000001111;
66 : 00111010011101111000;
67 : 00111010111010011000;
68 : 00111011010101101111;
69 : 00111011101111111100;
```

```
70 : 00111100001000111110;
71 : 00111100100000110110;
72 : 00111001101111100001;
73 : 00111101001101000001;
74 : 00111101100001010100;
75 : 00111101110100011011;
76 : 00111110000110010101;
77 : 00111110010111000001;
78 : 00111101100110011111;
79 : 00111101101001011111;
80 : 00111111000001110001;
81 : 00111111001101100100;
82 : 00111111011000001000;
83 : 00111111100001011110;
84 : 00111111101001100011;
85 : 00111111110000011010;
86 : 00111111110110000001;
87 : 00111111111010011000;
88 : 00111111111101100000;
89 : 00111111111111011000;
90 : 01000000000000000000;
91 : 00111111111111011000;
92 : 00111111111101100000;
93 : 00111111111010011000;
94 : 00111111110110000001;
95 : 00111111110000011010;
96 : 00111111101001100011;
97 : 00111111100001011110;
98 : 00111111011000001000;
99 : 00111111001101100100;
100 : 00111111000001110001;
101 : 00111110110100101111;
102 : 00111110100110011111;
103 : 00111110010111000001;
104 : 00111110000110010101;
105 : 00111101110100011011;
106 : 00111101100001010100;
107 : 00111101001101000001;
108 : 00111100110111100001;
109 : 00111100100000110110;
110 : 00111100001000111110;
111 : 00111011101111111100;
112 : 00111011010101101111;
113 : 00111010111010011000;
114 : 00111010011101111000;
115 : 00111010000000001111;
116 : 00111001100001011101;
117 : 00111001000001100100;
118 : 00111000100000100011;
```

```
119 : 00110111111110011100;
120 : 00110111011011001111;
121 : 00110110110110111101;
122 : 00110110010001100110;
123 : 00110101101011001100;
124 : 00110101000011101111;
125 : 00110100011011001111;
126 : 00110011110001101110;
127 : 00110011000111001101;
128 : 00110010011011101100;
129 : 00110001101111001100;
130 : 00110001000001101101;
131 : 00110000010011010010;
132 : 00101111100011111010;
133 : 00101110110011100111;
134 : 00101110000010011010;
135 : 00101101010000010011;
136 : 00101100011101010100;
137 : 00101011101001011101;
138 : 00101010110100110000;
139 : 00101001111111001101;
140 : 00101001001000110110;
141 : 00101000010001101100;
142 : 00100111011001101111;
143 : 00100110100001000010;
144 : 00100101100111100100;
145 : 00100100101101010111;
146 : 00100011110010011101;
147 : 00100010110110110101;
148 : 00100001111010100011;
149 : 00100000111101100110;
150 : 00011111111111111111;
151 : 00011111000001110001;
152 : 00011110000010111101;
153 : 00011101000011100010;
154 : 00011100000011100100;
155 : 00011011000011000010;
156 : 00011010000001111111;
157 : 00011001000000011011;
158 : 00010111111110011000;
159 : 00010110111011111000;
160 : 00010101111000111010;
161 : 00010100110101100001;
162 : 00010011110001101110;
163 : 00010010101101100011;
164 : 00010001101001000000;
165 : 00010000100100000111;
166 : 00001111011110111010;
167 : 00001110011001011001;
```

```
168 : 00001101010011100110;
169 : 00001100001101100011;
170 : 00001011000111010000;
171 : 00001010000000110000;
172 : 00001000111010000011;
173 : 00000111110011001011;
174 : 00000110101100001001;
175 : 00000101100100111111;
176 : 00000100011101101110;
177 : 00000011010110010111;
178 : 00000010001110111100;
179 : 00000001000111011111;
180 : 00000000000000000000;
END;
```

cos_lookup.mif

```
WIDTH=20;
DEPTH=181;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;

CONTENT BEGIN
0 : 01000000000000000000;
1 : 00111111111111011000;
2 : 00111111111101100000;
3 : 00111111111010011000;
4 : 00111111110110000001;
5 : 00111111110000011010;
6 : 00111111101001100011;
7 : 00111111100001011110;
8 : 00111111011000001000;
9 : 00111111001101100100;
10 : 00111111000001110001;
11 : 00111110110100101111;
12 : 00111110100110011111;
13 : 00111110010111000001;
14 : 00111110000110010101;
15 : 00111101110100011011;
16 : 00111101100001010100;
17 : 00111101001101000001;
18 : 00111100110111100001;
19 : 00111100100000110110;
20 : 00111100001000111110;
21 : 00111011101111111100;
22 : 00111011010101101111;
23 : 00111010111010011000;
```

```
24 : 00111010011101111000;
25 : 00111010000000001111;
26 : 00111001100001011101;
27 : 00111001000001100100;
28 : 00111000100000100011;
29 : 00110111111110011100;
30 : 00110111011011001111;
31 : 00110110110110111101;
32 : 00110110010001100110;
33 : 00110101101011001100;
34 : 00110101000011101111;
35 : 00110100011011001111;
36 : 00110011110001101110;
37 : 00110011000111001101;
38 : 00110010011011101100;
39 : 00110001101111001100;
40 : 00110001000001101101;
41 : 00110000010011010010;
42 : 00101111100011111010;
43 : 00101110110011100111;
44 : 00101110000010011010;
45 : 00101101010000010011;
46 : 00101100011101010100;
47 : 00101011101001011101;
48 : 00101010110100110000;
49 : 00101001111111001101;
50 : 00101001001000110110;
51 : 00101000010001101100;
52 : 00100111011001101111;
53 : 00100110100001000010;
54 : 00100101100111100100;
55 : 00100100101101010111;
56 : 00100011110010011101;
57 : 00100010110110110101;
58 : 00100001111010100011;
59 : 00100000111101100110;
60 : 00100000000000000000;
61 : 00011111000001110001;
62 : 00011110000010111101;
63 : 00011101000011100010;
64 : 00011100000011100100;
65 : 00011011000011000010;
66 : 00011010000001111111;
67 : 00011001000000011011;
68 : 00010111111110011000;
69 : 00010110111011111000;
70 : 00010101111000111010;
71 : 00010100110101100001;
72 : 00010011110001101110;
```

```
73 : 0001001010110110011;
74 : 0001000110100100000;
75 : 0001000010010000111;
76 : 0000111101111010;
77 : 0000111001100101001;
78 : 0000110101001110011;
79 : 0000110000110110011;
80 : 0000101100011101000;
81 : 0000101000000110000;
82 : 0000100011101000011;
83 : 0000011111001101011;
84 : 0000011010110000100;
85 : 0000010110010011111;
86 : 0000010001110110111;
87 : 0000001101011001011;
88 : 0000001000111011100;
89 : 0000000100011101111;
90 : 0000000000000000000;
91 : 1111111011100010001;
92 : 1111110111000100100;
93 : 1111110010100110100;
94 : 1111101110001001010;
95 : 1111101001101100001;
96 : 1111100101001111011;
97 : 1111100000110011010;
98 : 1111011100010111101;
99 : 1111010111111101000;
100 : 1111010011100011000;
101 : 1111001111001001101;
102 : 1111001010110001101;
103 : 1111000110011010011;
104 : 1111000010000100011;
105 : 1110111101101111001;
106 : 1110111001011100000;
107 : 1110110101001001101;
108 : 1110110000111001010;
109 : 1110101100101001111;
110 : 1110101000011100011;
111 : 1110100100010000100;
112 : 1110100000000110100;
113 : 1110011011111110010;
114 : 1110010111111000001;
115 : 1110010011110011110;
116 : 1110001111110001100;
117 : 1110001011110001110;
118 : 1110000111110100011;
119 : 1110000011111000111;
120 : 1100000000000000001;
121 : 1101111100001001101;
```

```
122 : 11011110000101011101;
123 : 11011101001001001011;
124 : 11011100001101100011;
125 : 11011011010010101001;
126 : 11011010011000011100;
127 : 11011001011110111110;
128 : 11011000100110010001;
129 : 11010111101110010100;
130 : 11010110110111001010;
131 : 11010110000000110011;
132 : 11010101001011010000;
133 : 11010100010110100011;
134 : 11010011100010101100;
135 : 11010010101111101101;
136 : 11010001111101100110;
137 : 11010001001100011001;
138 : 11010000011100000110;
139 : 11001111101100101110;
140 : 11001110111110010011;
141 : 11001110010000110100;
142 : 11001101100100010100;
143 : 11001100111000110011;
144 : 11001100001110010010;
145 : 11001011100100110001;
146 : 11001010111100010001;
147 : 11001010010100110100;
148 : 11001001101110011010;
149 : 11001001001001000011;
150 : 11001000100100110001;
151 : 11001000000001100100;
152 : 11000111011111011101;
153 : 11000110111110011100;
154 : 11000110011110100011;
155 : 11000101111111110001;
156 : 11000101100010001000;
157 : 11000101000101101000;
158 : 11000100101010010001;
159 : 11000100010000000100;
160 : 11000011110111000010;
161 : 11000011011111001010;
162 : 11000011001000011111;
163 : 11000010110010111111;
164 : 11000010011110101100;
165 : 11000010001011100101;
166 : 11000001111001101011;
167 : 11000001101000111111;
168 : 11000001011001100001;
169 : 11000001001011010001;
170 : 11000000111110001111;
```

```
171 : 11000000110010011100;
172 : 11000000100111111000;
173 : 11000000011110100010;
174 : 11000000010110011101;
175 : 11000000001111100110;
176 : 11000000001001111111;
177 : 11000000000101101000;
178 : 11000000000010100000;
179 : 11000000000000101000;
180 : 11000000000000000000;
END;
```