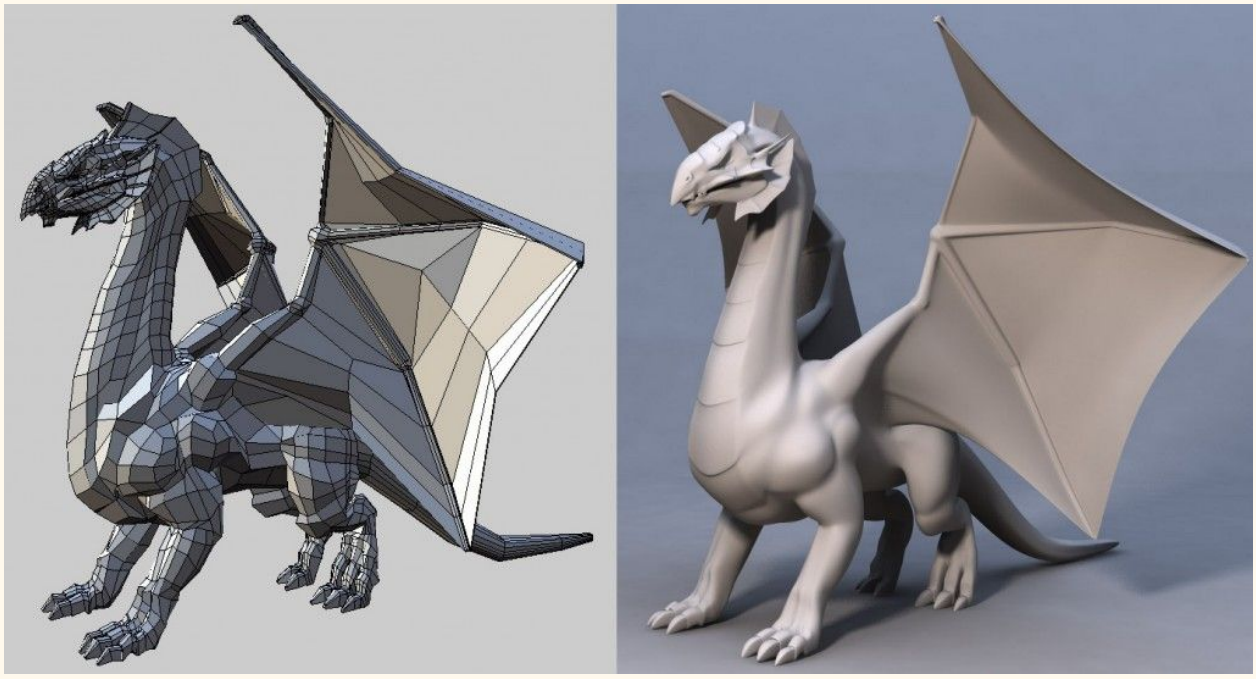
EECS 4840
Embedded System

3D

Graphics Accelerator

Jie Huang (jh4000), Chao Lin (cl3654), Zixiong Liu (zl2683), Kaige Zhang(kz2325)



INTRODUCTION

We plan to make a graphics accelerator that renders 3D models on the screen. The software will supply information of a 3D scene and the hardware will render the scene onto the screen. In the 3D model, each surface is considered as a collection of triangles that are defined by a set of vertices. Therefore, we will render in real time the given 3D model triangle by triangle, considering also its brightness and shading.

SYSTEM OVERVIEW

The user input will be triangle vertices in 3D coordinates, the color of these vertices, a model-view-projection matrix as well as a lighting vector and its color. The FPGA will contain a rendering pipeline, which will render the user input to a framebuffer on the off-chip SDRAM. A VGA output module will then send the framebuffer line by line to the VGA controller.

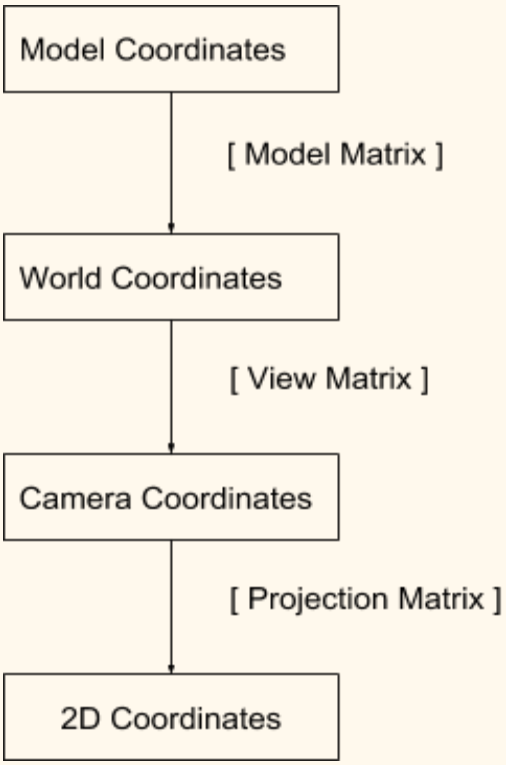
Software Overview

The role of the software is mainly to supply rendering information to the hardware. These information include:

- A vector of 3D coordinates (x,y,z) used to render 3D objects such that they are composed of triangles
- Model matrix composed of geometric transformation matrices
- View matrix that controls the viewpoint of 3D objects
- Projection matrix that converts 3D objects to 2D
- Color vector containing RGB values
- Lighting vector and its color vector

Library: OpenGL Math (GLM) is used to generate the matrices.

Model, View and Projection Matrix Overview



Model Matrix

Model matrix contains every geometric transformation applied to a given object; therefore, it is simply a matrix resulted from multiplying one or more transformation matrices, such as scaling, translation, and rotation, together.

Rotation in x, y, z directions

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; R_y = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; R_z = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

View Matrix

View matrix is used to simulate a moving camera view of our 3D objects. Multiplying world coordinates to the view matrix would convert the world space into camera space, which allows us to view the 3D object in various angles and directions.

$$\begin{bmatrix} right_x & up_x & forward_x & position_x \\ right_y & up_y & forward_y & position_y \\ right_z & up_z & forward_z & position_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection Matrix

There are two types of projection matrices: orthogonal and perspective. For this project, orthogonal projection matrix will be used and is structured as follows:

$$P = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

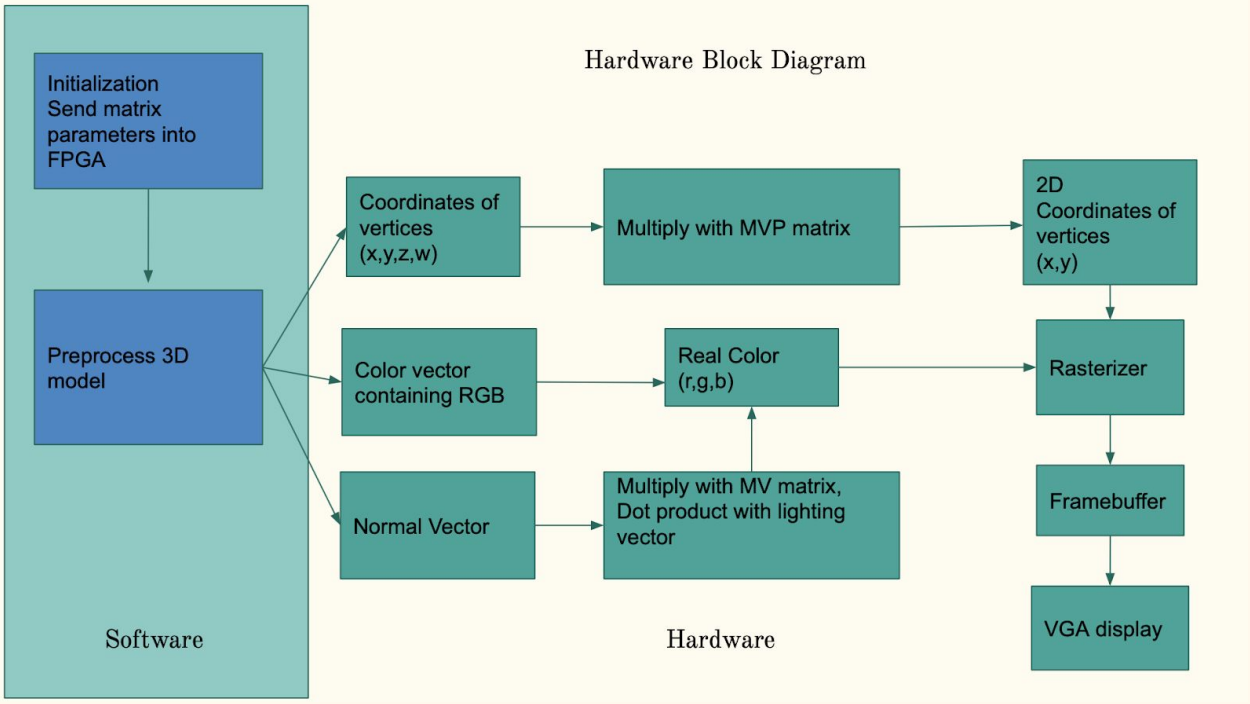
where right, left, top, bottom are positions of the clipping plane.

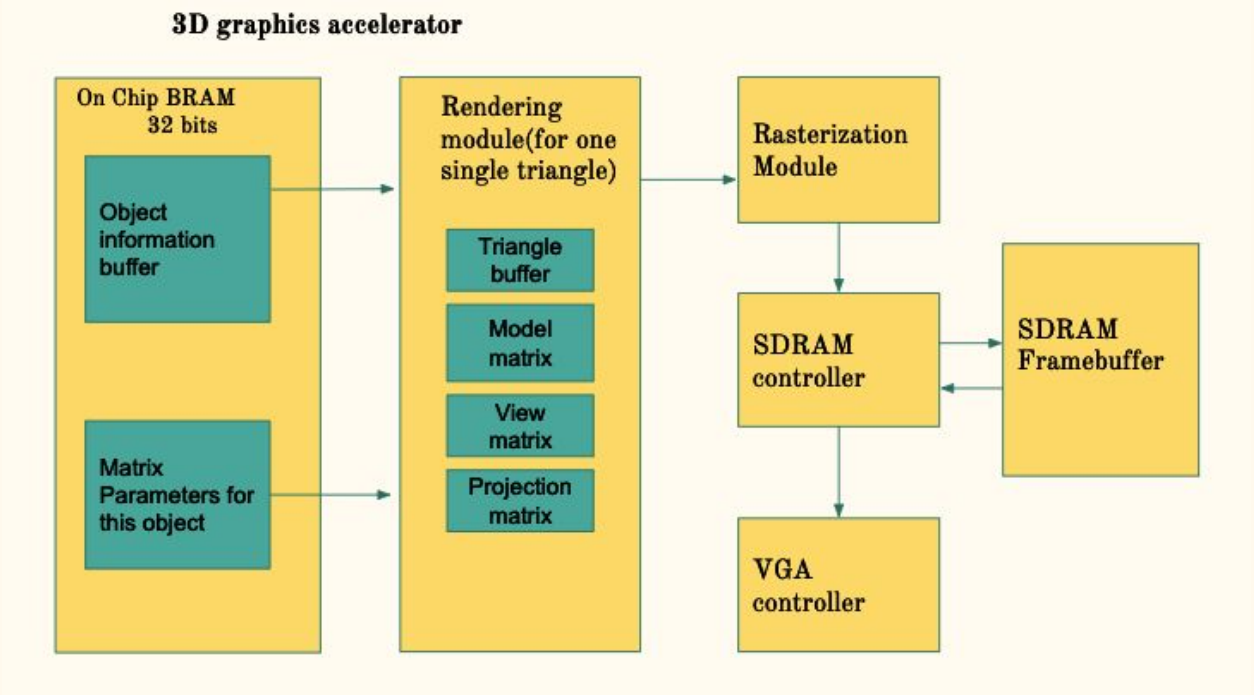
Hardware Overview

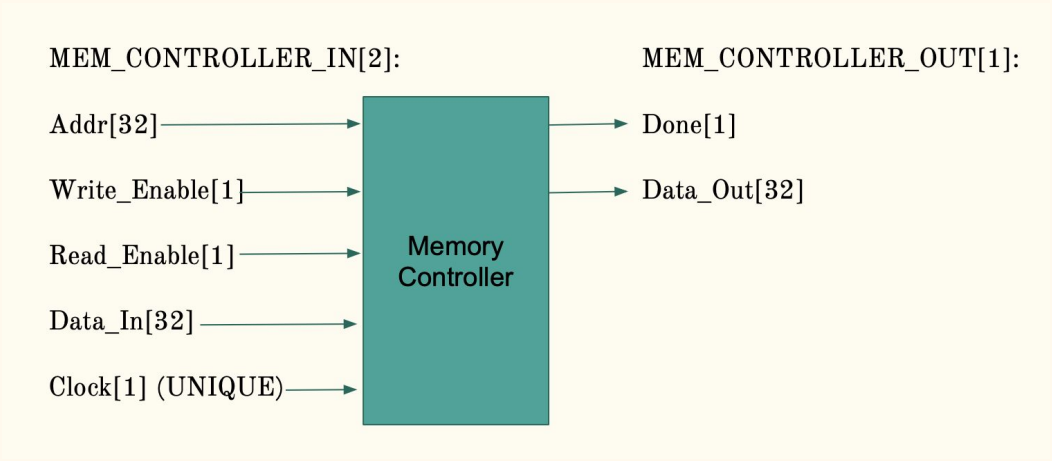
Our hardware contains a memory I/O to read all necessary data from Avalon Bus. Then we design the FPGA to behave as a shader which basically fills the triangles defined by input coordinates, color vectors, lighting vectors and Model, View and Projection matrices. After computation, the information will firstly be saved as framebuffer to feed VGA display.

The details of the computations are as follows:

(1) Computation of 2D coordinates and depth: the homogeneous 3D coordinate of each vertex is multiplied to the MVP matrix, which is a linear mapping from the model space to the projection space, to obtain the 2D coordinates on the screen plus depth.
(2) Rasterization: Having obtained the 2D coordinates and the color of the vertices, the triangle is then filled by interpolation the three vertices' color.
(3) Computation of real color: the normal vector of each triangle of an object is supplied to the hardware, the hardware then multiply the normal vector with the MV matrix, yielding the normal vector in the world space. The intensity of color is produced from the dot product of the light direction vector (in world space coordinates) and the normal vector in the world space.
(4) The intensity is then clipped between 0 and 1 so that there is no negative intensity. The color produced by rasterization is then multiplied with this intensity to obtain the real color on the screen.
(5) Depth test: Having obtained the projection space coordinates (x, y, h) and color (r, g, b), the hardware first reads the framebuffer at point (x, y) to obtain the current depth h', which is then compared to h. If h' is closer to the view than h (i.e. shallower), then we do not render this point. Otherwise we update the framebuffer at (x, y) to (r, g, b, h).


Hardware Block Diagram
Initialization
Send matrix parameters into FPGA
Preprocess 3D model
Coordinates of vertices (x,y,z,w)
Color vector containing RGB
Normal Vector
Multiply with MVP matrix
Real Color (r,g,b)
Multiply with MV matrix, Dot product with lighting vector
2D Coordinates of vertices (x,y)
Rasterizer
Framebuffer
VGA display
Software
Hardware


3D graphics accelerator
On Chip BRAM 32 bits
Object information buffer
Matrix Parameters for this object
Rendering module(for one single triangle)
Triangle buffer
Model matrix
View matrix
Projection matrix
Rasterization Module
SDRAM controller
SDRAM Framebuffer
VGA controller



MEMORY BUDGET

Each object has thousands of triangles, each triangle has the following data that should be sent to the SRAM on FPGA:

- Coordinates of 3 vertices (x,y,z,w), each coordinate has 32 bits(4 bytes):
  $4\ bytes \times 4 \times 3 = 48\ bytes$
- Color of 3 vertices (R,G,B, notused): $4\ bytes \times 3 = 12\ bytes$
- Normal vector (x,y,z,w) : $4 \times 4 = 16\ bytes$

In total it is 76 bytes

Different object has different matrix parameters

- Model matrix: 5 4×4 matrices $= 5 \times 4 \times 4 \times 4\ bytes = 320\ bytes$
- View matrix: $4 \times 4 \times 4\ bytes = 64\ bytes$
- Projection matrix: $4 \times 4 \times 4\ bytes = 64\ bytes$

In total, 448 bytes.

Framebuffer: $480 \times 640 \times 4 = 1.2288$ Mbytes

MILESTONES

Milestone 1 (April 5th):

- Implement memory controller
- Build the software to feed 3D model data

Milestone 2 (April 19th):

- Build rendering unit and rasterizer
- Software simulation for rendering correctness

Milestone 3 (May 3rd):

- Testing and Debugging

References

- https://solarianprogrammer.com/2013/05/22/opengl-101-matrices-projection-view-model/
- http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/