# 3D PONG


## Igor Ginzburg

Introductory Digital Systems Laboratory
5/14/2006



***Abstract***

*3D Pong takes MIT Pong to the next level with a 3D interface. At the heart of the project there is a hardware based 3D renderer. The renderer takes in a 3D model, specifically a sequence of colored triangles in a 3D space, and produces a 2D VGA image. The renderer supports arbitrary translations, rotations, applies flat shading, and uses orthogonal projection. The renderer's data path is composed of a Triangle Source which extracts the models from on-chip ROM, a Triangle Pipeline which applies a series of transformations to each triangle, a Triangle Shader which rasterizes the triangles, and a Screen Buffer which buffers the generated frames in off-chip RAM before they are sent to a VGA monitor. While the game play of 3D pong is identical to MIT Pong, the game field is game field is made up of 3D objects. The game is visually pleasing and fun to play.*

**Contents**



**List of Figures**

## 1. Overview

Pong, developed by Atari in 1972, was the first popular video game. The game is a simple simulation of table tennis, or ping pong. It was played in arcades and later on home televisions. The game logic was implemented in digital hardware, consisting mostly of 2-input NANDs on TI's SN7400 4-NAND chips. In the Fall of 2005, MIT Pong, a game in the spirit of the original Atari game, was added to the MIT 6.111 curriculum. This paper takes pong one step further, proposing 3D Pong, a game with a 3D display. Screenshots of the three different games, can be seen in Figure 1 in chronological order.

Image removed due to copyright restrictions.
Please see, for example, http://en.wikipedia.org/wiki/Image:Pong.png

*Figure 1: Game Field Screenshots: Atari Pong, MIT Pong, 3D Pong*

The game play for 3D pong is identical to MIT Pong. The player controls one rectangular paddle, which is oriented vertically and placed on the left edge of the game field. The paddle can be moved up and down on the game field through up and down buttons on the lab kit. Borders are placed at the top, right, and bottom edges of the screen. The game is played with one ball, which moves inertially within the screen, or game field. When the ball strikes a border, it bounces off at the same angle it came in. Collisions with the paddle are more complicated, in an effort to make the game more difficult. If the ball strikes the left edge of the game field, the ball freezes, the game is over and the player has lost. There is no way to win, but the game can last indefinitely, providing continuous entertainment. Three switches control the initial velocity of the ball, specifying the speeds both in the horizontal and vertical direction.

Another three switches control the sensitivity of the paddle buttons. Greater sensitivity allows the paddle to outrun the ball, but makes precise positioning of the paddle more difficult. While the initial velocity of the ball only matters on reset, the paddle sensitivity can be changed as the game progresses.

What makes 3D Pong different from its predecessors is its use of a hardware based 3D renderer. The renderer is at the heart of the project. Its design is very general, allowing it to project arbitrary 3D models onto a 2D VGA display. For example, in Figure 2, we see the renderer used to project a helicopter under different rotations.

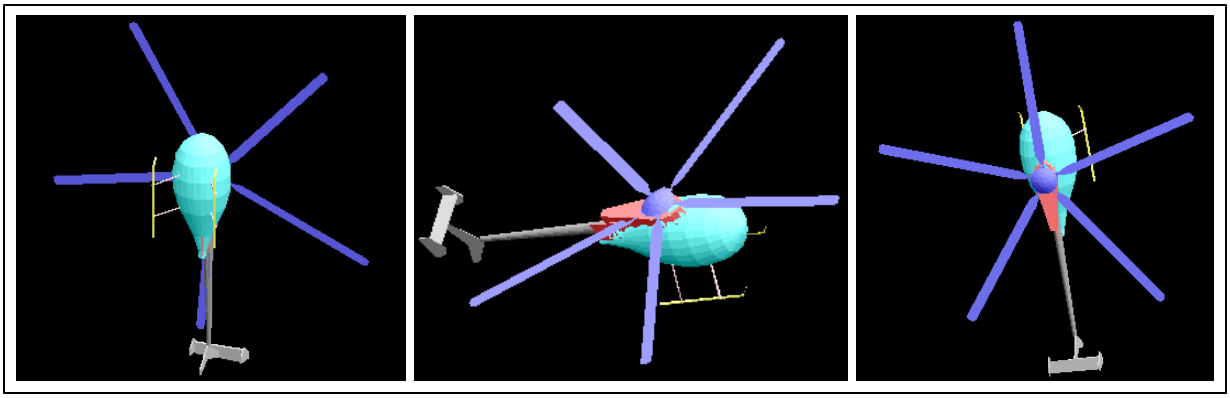

*Figure 2: Renderer Screenshots*

The renderer is feature rich yet simple. It can apply arbitrary 3D rotations and translations, both to entire models, and to individual objects within the model. Flat shading is used to simulate lighting effects from a virtual light source. The models can be resized through a wide range of magnifications. Care is taken so that obstructions in the field of view are rendered properly. This feature set was selected, because it creates a realistic 3D effect, while allowing for simple hardware. This can be seen in the fact that the implementation of the renderer does not use any division.

There are several external inputs to the renderer. A switch is used to select whether the pong game field or the helicopter model is displayed. A set of six buttons and a switch is used to

input arbitrary rotations and translations which the renderer applies to either model in real time. A global reset button is used to restart the game and place the model into a default orientation.

## 2. Description

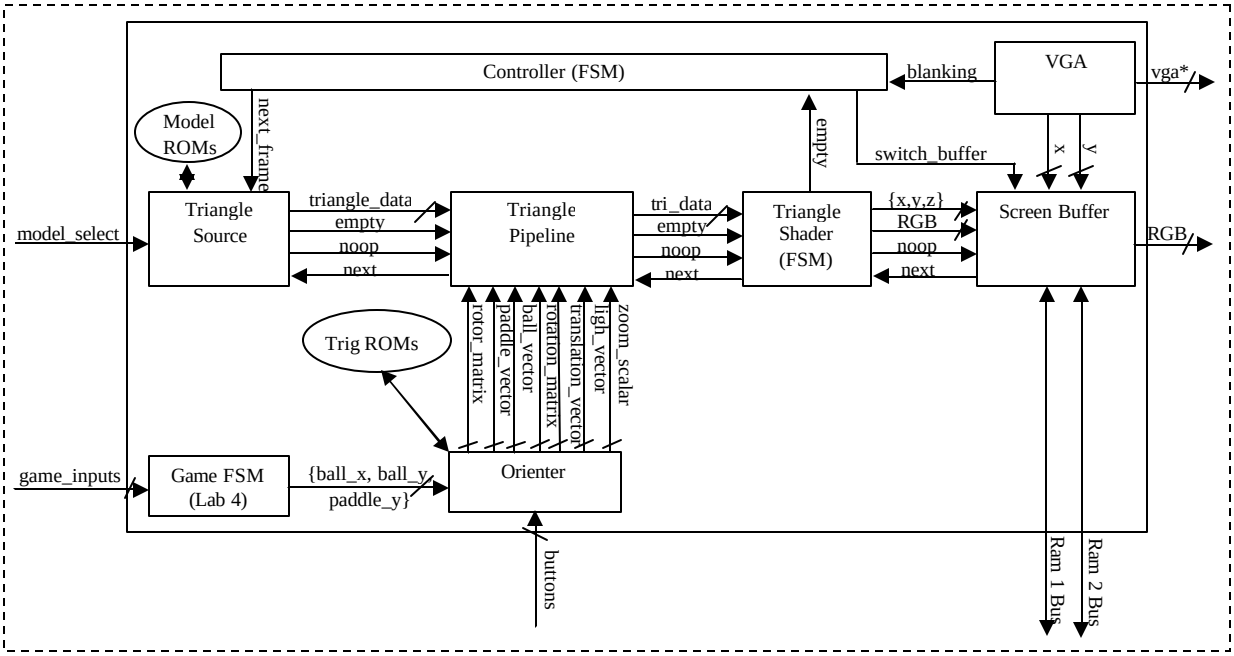


*Figure 3: 3D Pong Block Diagram*

3D Pong consists of eight main subsystems. These subsystems are further divided into a total of 62 Verilog modules. Four of these subsystems, in particular the Triangle Source, Triangle Pipeline, Triangle Shader, and Screen Buffer, comprise the data path for the renderer. The remaining four subsystems, including the Game Fsm, the Orienter, the Controller FSM, and the VGA driver, provide control signals for the data path.

### 2.1 Data Path Subsystems

The Triangle Source subsystem loads a 3D model, one triangle at a time from, from a ROM module. The triangles are sent out the *triangle_data* output to the Triangle Pipeline, as requested through the *next* input. When there is no triangle ready for the current clock cycle, the *noop* output is asserted, and the *triangle_data* output is ignored. If all the triangles that make a model have been outputted, the *empty* output is asserted. These handshaking signals (*empty*,

*noop*, and *next*) are used throughout the modules on the data path to coordinate data flow. The *next_frame* input from the Controller FSM determines when to go back to outputting the first triangle in the model. The *model_select* input determines which of the two models, game or helicopter, to output.

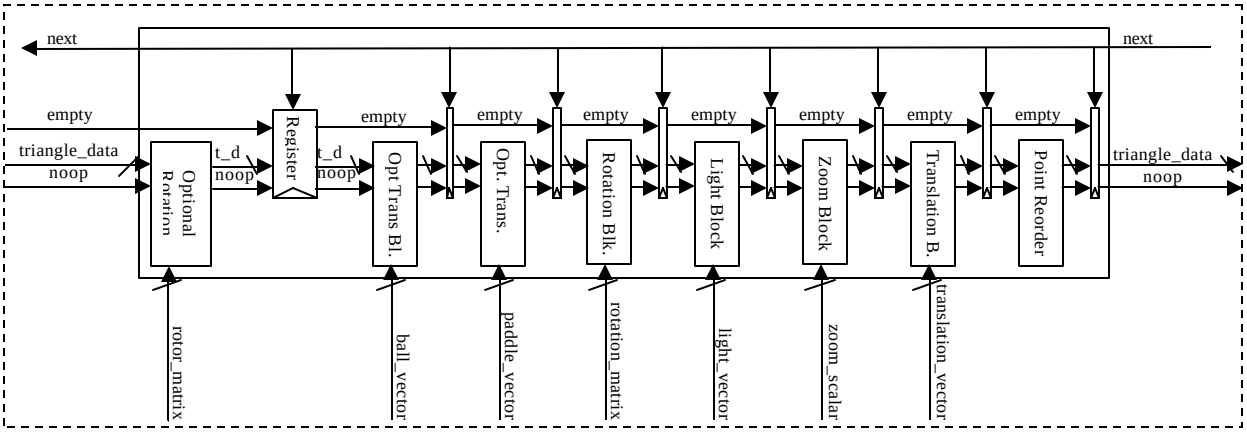


*Figure 4: Triangle Pipeline Block Diagram*

The Triangle Pipeline puts the triangles emerging out of the Triangle Source through a series of transformations. These include rotations, translations, shading, rescaling, and point reordering. These transformations are controlled by the *rotor_matrix*, *paddle_vector*, *ball_vector*, *rotation_matrix*, *translation_vector*, *ligh_vector*, and *zoom_scalar* inputs originating in the Orienter subsystem. The transformed triangles are sent out to the Triangle Shader subsystem. The default handshaking signals, in particular *empty*, *noop*, and *next*, are used to coordinate data transfer between the Triangle Pipeline and the Triangle Shader.

The Triangle Shader rasterizes the triangles presented by the Triangle Pipeline. For each triangle, the set of points on the screen that fall within the triangle is sent one at a time to the Screen Buffer subsystem. The handshaking signals between the Screen Buffer and the Triangle Pipeline are limited to just *noop* and *next* signals. An *empty* signal is however sent to the Controller FSM when the Triangel Shader. So, the empty signal, which originates at the

Traingle Source does not reach the Controller FSM until all triangles in the system have been rasterized.

The Screen Buffer subsystem maintains two screen buffers in two off-chip ZBT SRAMs. A pixel stored in the screen buffer takes up one 36-bit word in the SRAM. In addition to 24 bits of RGB color, a 12 bit z-coordinate is stored for each pixel. This z-coordinate , or z-buffer, is used to correctly render triangles that overlap in the x-y plane. Objects with higher z coordinates are assumed to be farther away. Therefore, if the screen buffer is given pixel data with a higher z-coordinate then the current data stored for the pixel, the new data is ignored. This scheme requires a read of the old z-coordinate prior to the write. So, two cycles are required to place one pixel in the screen buffer.

During 3D Pong operation, one SRAM is used for storing pixels produced by the Triangle Shader, while the other is used to generate the *rgb* output sent to the DAC which drives the VGA output. When the rendering of a new frame begins, the *switch_buffer* input of the screen buffer is asserted. The two SRAMs switch rolls. The SRAM used for creating the *rgb* output can begin working immediately. The contents of the other SRAM must be reset before new pixels can be written.

**2.2 Control Subsystems**

The Controller FSM acts as a major FSM. Its job is to decide when to switch the screen buffers and begin working on the next frame. The Controller FSM's *empty* input is asserted by the Triangle Shader FSM when there are no more triangles to rasterize. The *blanking* input is asserted by the VGA driver during the vertical sync phase. The Control FSM's goal is to keep the pipeline as busy as possible without switching frames during the vertical active video VGA phase. This goal is accomplished by following the transition diagram shown below in figure 5.

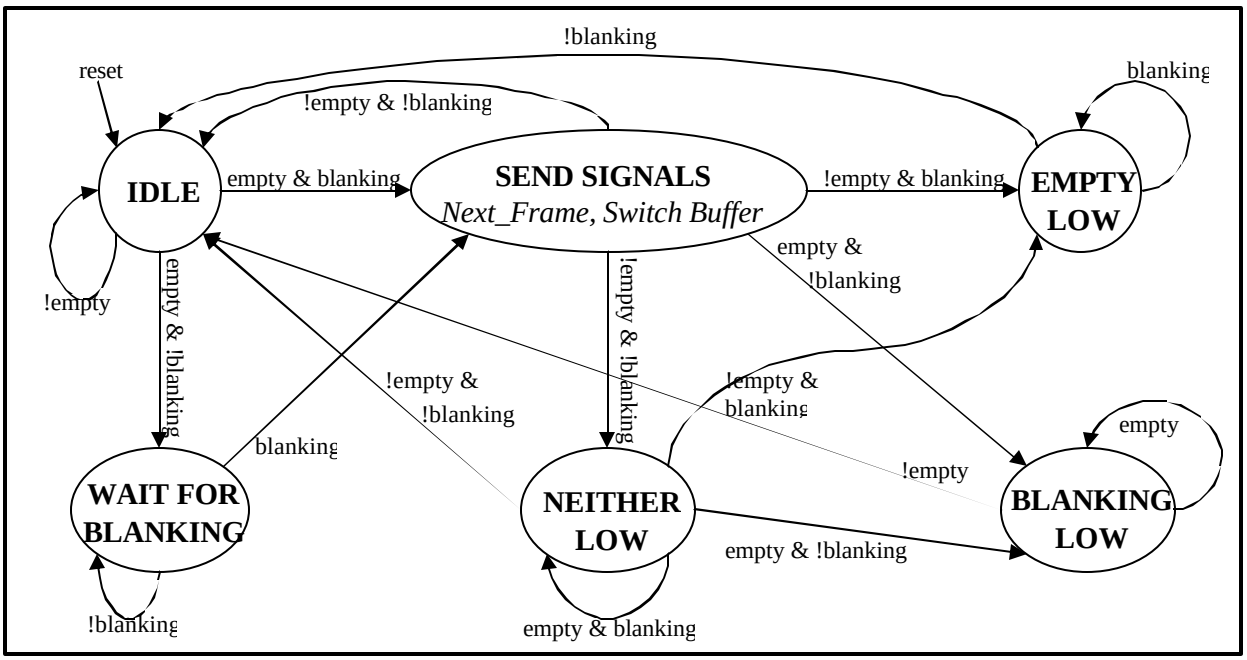


*Figure 5: Controller FSM State Transition Diagram*

The Orienter subsystem generates control signals for the Triangle Pipeline. These include several rotation matrices, translation vectors, and scalars meant for zoom. These outputs are based on a combination of inputs, including external buttons and switches, along with position inputs generated by the Game FSM. In order to keep the system responsive, these inputs are sampled at a constant rate (equal to the VGA refresh rate). In order to make sure that all triangles rendered in the same frame are renderer based on the same control signals, the Orienter buffers the intermediate control signals, only changing its outputs when the *next_frame* signals from the Controller FSM is asserted.

The vector and scalar control signals can be easily generated by accumulators which rely on some signed addition. The rotation matrices are more difficult to generate. Accumulators can be used to determine, in degrees or radians, an angle of revolution about each of the three axes. To create a rotation matrix about one axis, the controller uses a Trig Coregen Module which calculates the sine and cosine of the angle. The controller calculates three intermediate rotation

matrices, one per each axis, and multiplies them together to come up with one rotation matrix output.

The Game FSM contains the logic for MIT Pong and is identical to my Lab 4 Game FSM. The paddle's vertical position and the ball's position in the x/y plane are passed to the Controller FSM, which in turns them into translation vectors. Several of the parameters to the Game FSM are tweaked to make the paddle and ball easier to see on a 3D game field.

The VGA module is very similar to its Lab 4 counterpart. Its parameters have been changed to match the 60hz VGA refresh rate as apposed to the 75hz refresh rate in Lab 4. This compensates for a reduction in the pixel clock frequency, allowing for more complex combinational logic in a single pipeline stage. The VGA module produces an additional Blanking output which is used by the Controller FSM.

**3. Implementation and Testing**

The implementation of 3D Pong began with a software mock-up written in Java. The mock up was very useful during the design phase, in determining the structure of the renderer. Several features in the original mock-up, like perspective rendering, which makes objects farther away appear smaller, were removed to simplify the hardware implementation. The mock-up was also used in the design of the models, since compiling all the different iterations of the models into the ROMs of a hardware renderer would have taken an unreasonable amount of time. The mock-up, along with an excel spreadsheet, were used to generate the input ".coe" files used to preload the on-chip ROMs.

The data path subsystems were implemented before the control subsystems. While the initial implementation was tested using the ModelSim simulator, a logic analyzer was used to debug the interactions of the Screen Buffer with the ZBT SRAMs.

Several problems emerged during the implementation. The VGA output displayed on the LCD Monitor was very poor in quality. Even for small models covering a small portion of the display, the black background would be full of little artifacts. These included pixels and sub-pixels lit in colors not present in the model. After a lot of trial and error, this problem was fixed by placing a DCM between the renderer's internal clock and the pixel clock sent to the VGA. The assumption is that the internal clock signal may be degraded by the capacitance on the wires sending the clock to the ZBT SRAMS. While the DCM does not change the clock frequency, it should regenerate the clock signal, producing a quality square wave. It is possible that either the DAC creating the analog VGA output or the LCD Monitor requires a well formed clock.

**4. Analysis and Conclusion**

There are several bottlenecks that limit the frame rate of the renderer. The biggest one is the throughput the Screen Buffer gets to the ZBT SRAMs. Writing every pixel requires two clock cycles, and clearing the buffer between frames takes a large amount of time. The throughput can be improved by clocking the SRAMs and the Screen Buffer code at a higher clock rate than the rest of the pipeline. A global reset for the SRAM would be very helpful in shortening the time spent on clearing the buffer. In the absence of these improvements, an on-chip circular pixel buffer, either in registers or on-chip RAM, could be useful for eliminating no-ops from the pipeline. This simple modification would be able to double the frame rate for models that take up a large proportion of the screen.

The implementation of 3D Pong is visually pleasing and fun to play. It is a worthy successor to MIT Pong and the original Atari game. The amount of work entailed in implementing a hardware renderer, as apposed to the software mock-up, has given me a great appreciation for the work of digital designers.