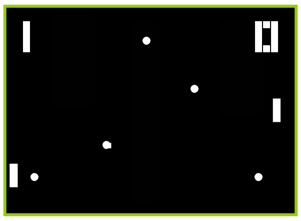
Three-Dimensional Ray-Cast Pong

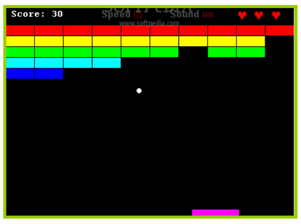
Richard Hughes
Elizabeth Power!

Overview

- **What is Pong?**
  - Traditional Pong is a two-dimensional game that "simulates" table tennis. The player controls a paddle by moving it vertically on the left side of the screen to block the ball.
  - The goal is for the player to keep the ball from hitting the left side of the screen.

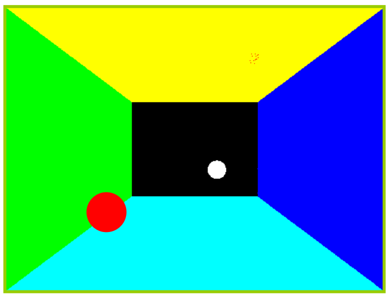
- **Common variations…**
  - 2 Player
    - Compete against a computer controlled opponent or another player which controls a second paddle on the right side of the screen
  - Multiple Balls
    - Added complexity of having to keep track of 2 to 5 balls
  - *Pong Doubles and Elimination (aka Quadrapong)*
    - Version for up to 4 Players
    - Each player starts with 4 points and looses one each time they miss the ball





  - Doctor Pong (aka Puppy Pong)
    - An adoption for use in a non-coin-operated environment
    - Specifically used to occupy kids in doctors' waiting rooms.
  - Breakout
    - Pong with bricks
    - A layer of bricks lines the top third of the screen and when the ball hits a brick, the ball bounces away and the brick is destroyed.

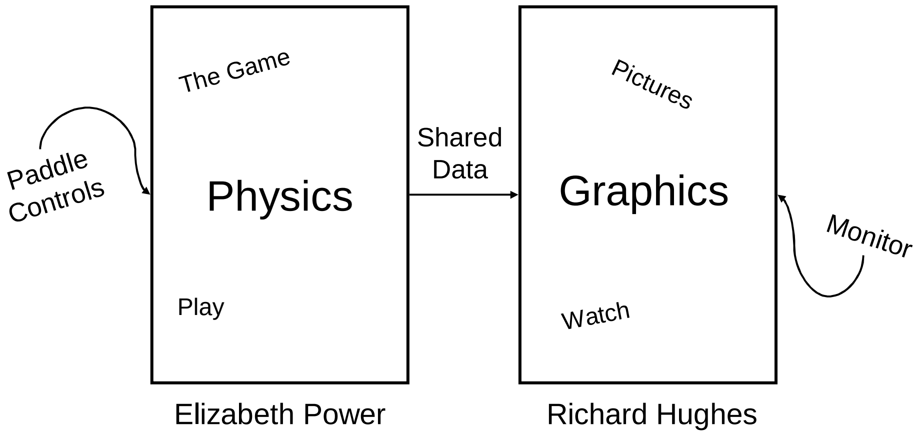
What Makes Our Pong Special?



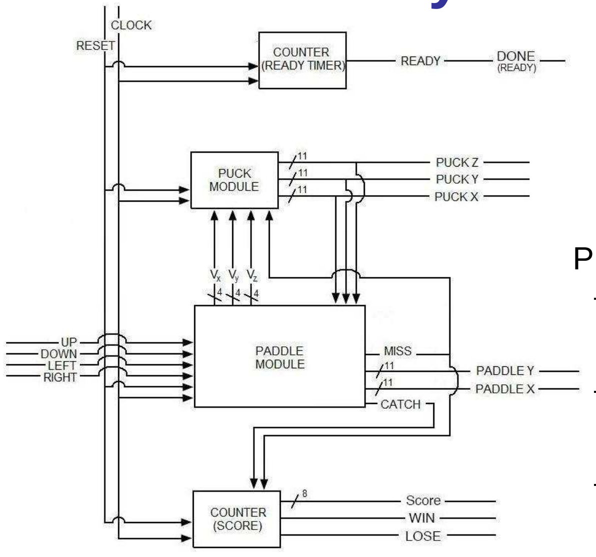
- 3-Dimensional
  - Functionally
    - Puck moves in x, y, and z directions
    - Paddle moves in x and y directions
  - Graphically
    - Walls are shaped and shaded to look more realistic
    - Puck is shaded to look like a sphere
    - Paddle shaded to have a elliptical surface (if time permits)
- Increasing complexity
  - Each time you catch the ball its speed increases
  - The reflection angle changes based on where on the paddle you catch the ball
  - Multiple balls (if time permits)
- Keeps score
  - Reach 63 to win

Winner!

Division of Labor



Physics



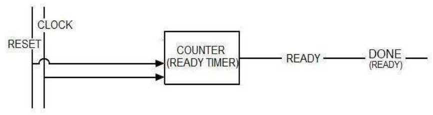
The physical implementation of our game.

Purpose:

- Manage interactions between Puck and Paddle
- Monitoring the score & status of the game instance
- Create a "Ready" signal for synchronization with Graphics Division

Physics: Ready

In order to reduce synchronization issues between the Physics & Graphic Divisions, the Physics Division will create a "Ready" signal for the Graphics to use.

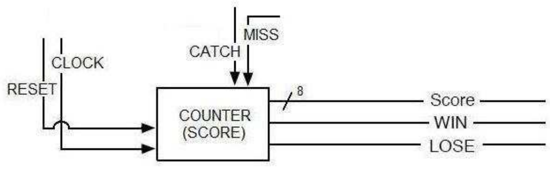


Physics: Score

- In
  - Catch
  - Miss
  - Clock
  - Reset
- Out
  - Score [7:0]
  - Win
  - Lose

The Score Module keeps track of the status of the game.

- It counts how many times the paddle "catches" the puck and outputs that as the score.
- It also decides and tells you when you win or lose.



Physics: Paddle

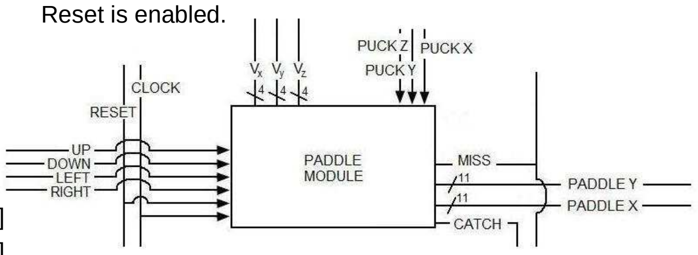
- In
  - Clock
  - Reset
  - Up
  - Down
  - Left
  - Right
  - Puck_x [11:0]
  - Puck_y [11:0]
  - Puck_z [11:0]
- Out
  - Miss
  - Catch
  - V_x [3:0]
  - V_y [3:0]
  - V_z [3:0]
  - Paddle_x [11:0]
  - Paddle_y [11:0]

The Paddle Module does all of the hard work.

- The Paddle Module take all of the game control inputs *(Up, Down, Left, Right)* and moves the Paddle accordingly.

- If the Paddle catches the Puck, the module will pulse "Catch." It will also increase the z velocity and change the x & y velocities based on where on the Paddle the Puck hits.

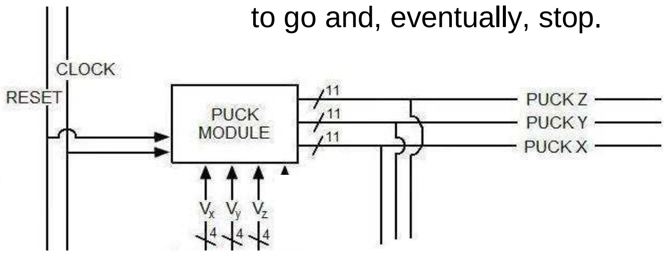
- If the Paddle misses the Puck, the module will enable "Miss" and the x, y, & z velocities will become 0 until Reset is enabled.



Physics: Puck

The only duty of the Puck Module is to go and, eventually, stop.

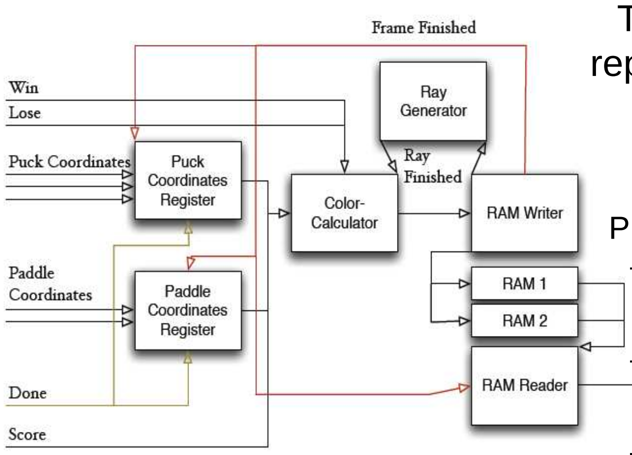
- In
  - Clock
  - Reset
  - V_x [3:0]
  - V_y [3:0]
  - V_z [3:0]
- Out
  - Puck_x [11:0]
  - Puck_y [11:0]
  - Puck_z [11:0]



- The Puck will go with the desired x, y, & z velocities until it hits a wall, at which time it will "bounce" off and continue with the properly inverted x, y, & z velocities.

- The Puck will stop all movement whenever the paddle misses the puck, indicating that the game is over. This is taken care of by the Paddle Module because it changes x, y, & z velocities to 0.

Graphics



The graphical representation of our game.

Purpose:

- Display interactions between Puck and Paddle
- Display the score & status of the game instance
- Render the graphics in 3 dimensions

Graphics: Implementation

- Ray-Casting
  - Phong is for suckers
  - Distance & Position functions are for winners
- *The Hard Part:* Division
  - 5x division for 5x planes
  - 1x division for ray-generation
  - 1x division for sphere intersection

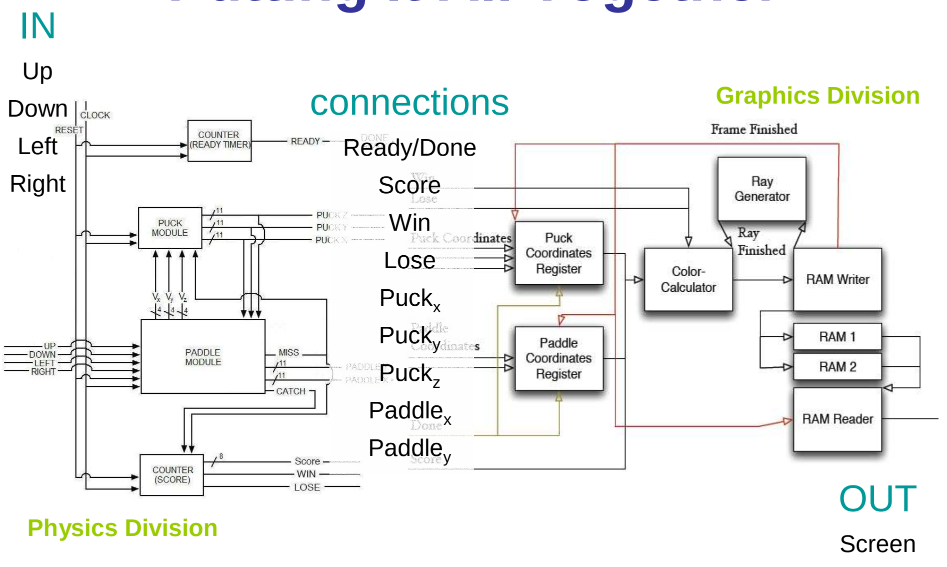
Graphics: Integer Math

- How much fixed-point precision?
  - Python tests indicate:
    - 10 bits is too few to accurately intersect the sphere
    - 12 bits is sufficient.
- How many bits do we need?
  - 12 bits 'to the right of the decimal point'
- How many bits left of the decimal point?
  - Multiplication makes enormous numbers
    - doubling # of bits
  - Keep the excess in registers in appropriate locations
  - Alter the precision with left-right shifts to keep scale appropriate.

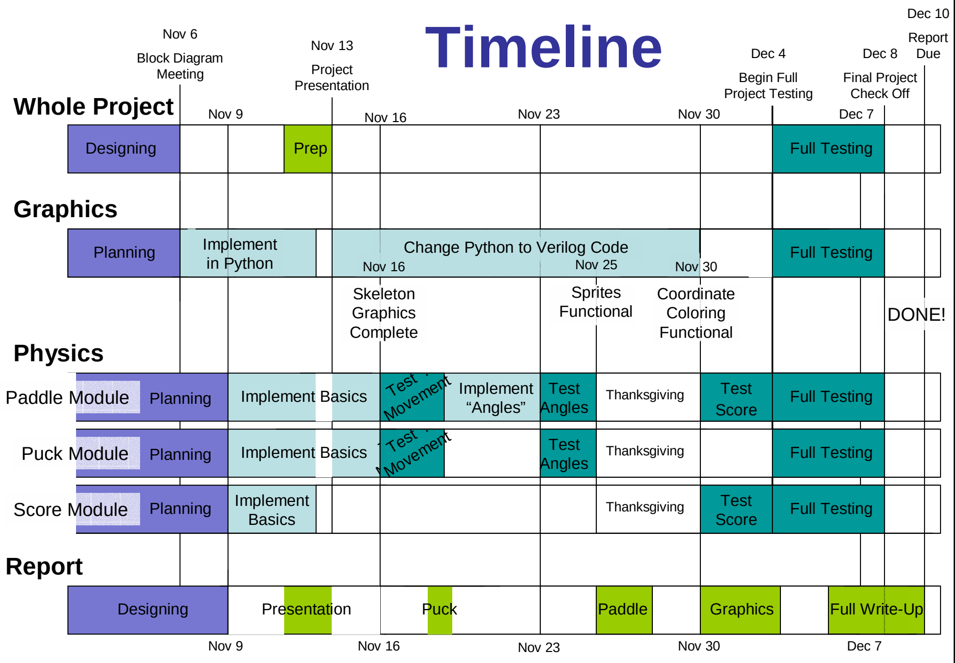
Graphics: Keeping things 3D

- Without phong shading, how do we ensure that I'm not faking it?
  - Coordinate-function coloring.
  - Color of plane: $P(x,y,z,s_X,s_Y,s_Z)$
  - Color of sphere: $S(s_X,s_Y,s_Z)$

Putting It All Together



Timeline



Testing Issues

- Testing Without the Other Division
  - Graphics Division
    - Use randomly generated values for Puck and Paddle locations and the Score
    - Use the clock to generate the ready signal
  - Physics Division
    - Use ModelSim to simulate parts of the game before the graphics are ready
- Modular Cooperation
  - Creation of “Skeleton” Graphs
    - Implemented by November 16
    - Provides Physics the ability to control the Paddle and visibly watch the Puck / Paddle interactions
  - Lengthy “Total Testing Time”
    - We have set aside almost a full week to ensure that the two divisions function properly together

Summary

- Complex Physics
- Ray-Casting Graphics
- VGA Resolution
- 30+ fps
- Mouse &/or Keyboard Support (if time permits)