BY Richard Hughes and Elizabeth Power **FOR** Chris Terman **IN** 6.111 **ON** Saturday, October 25, 2008

PROJECT PROPOSAL

Project Name: 3D Ray-trace Pong

Project Team Members: Elizabeth Power, Richard Hughes

In this project we will create a 3-d variant of of the pong project in lab 5 with ray-traced graphics. The game will have three dimensions of movement for the spherical puck and two dimensions of movement for the circular paddle, and it will keep track of your score (i.e., how many consecutive bounces you've managed) on-screen with hardware 'sprite' characters. The puck will bounce off the paddle at different angles depending on the current velocity and/or relative position of the paddle. The ray-tracing will include shadows, phong shading, checkerboard walls, and 8-bit color, with 3 bit red and green and 2 bit blue.

BY Richard Hughes and Elizabeth Power **FOR** Chris Terman **IN** 6.111 **ON** Saturday, October 25, 2008

Ray-Casting - Ray-Tracing For The Lazy

Ray-tracing is a graphics rendering algorithm that methodically renders a three-dimensional scene pixel-by-pixel. Each pixel corresponds to a single ray projected from a 'view point'; the rays are projected as if to intersect with the 'pixels' on an imaginary 'screen' defined in the 3D. Geometrical analysis is performed to determine what objects in the simulated 3D space each ray intersects; we perceive the closest intersection. We then perform geometric analysis to determine how much light each light-source casts on to that intersection surface by casting rays from the intersection to the light and determining the angle of incidence, how distant and bright the light is, and whether the light is shadowed. We sum this light, determine how much of it is being sent towards the screen, and that is the color of the pixel. We then repeat that for *every* pixel.

Technically, this process is called 'ray-casting', meaning that it is not iterative - ray-tracing is, technically, an iterative form of this process that uses rays cast off from the point of intersection to accurately model such effects as reflection, transparency, refraction, or shadows that blur with distance. We will not be attempting to model such effects as reflection or refraction and so ray-casting is sufficient for our needs. As a result, the mathematics required to calculate the intersection of a raw with any given geometry are *bounded*, and thus we can run each intersection-calculation in parallel without worrying about one taking a far longer time than another.

Our scene will include five planes and a sphere. In order to calculate where, and if, a ray intersects with a sphere, we need to calculate a square root of a fixed-point real number and perform division - both quite difficult in hardware. Calculating the intersection of a ray with a plane likewise requires division. Dividers take up a great deal of space and have a lot of latency, but good throughput. Thus, we'll want to find a way to use a single divider module for all intersections, so as to have efficient utilization of chip space.

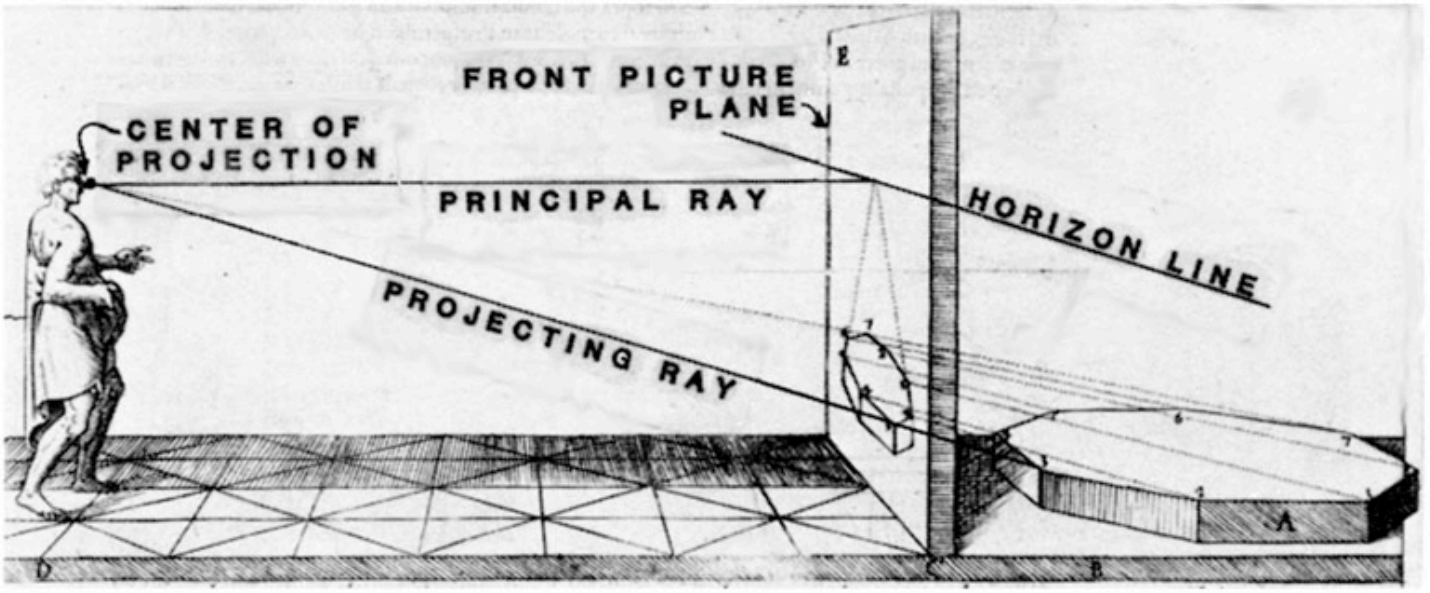
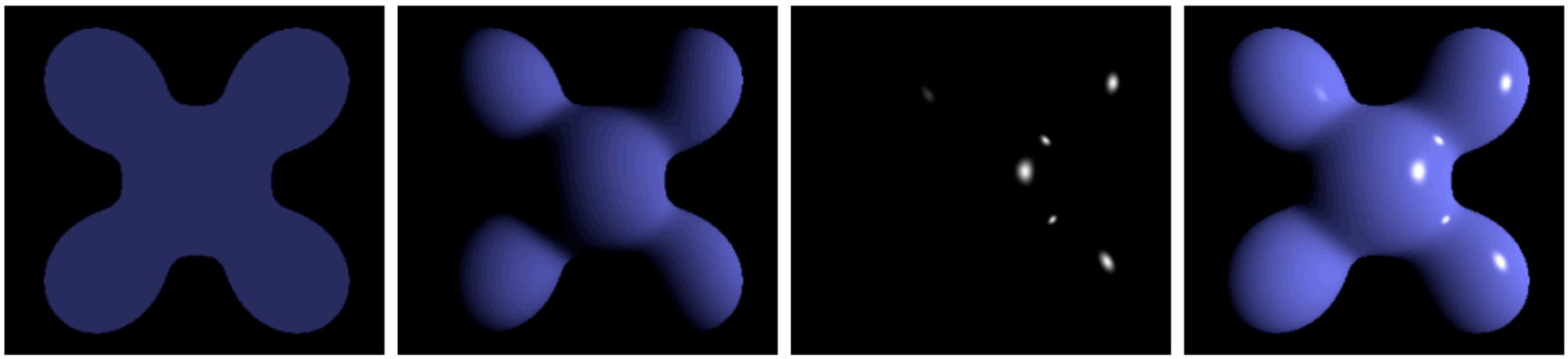


Figure 1: The 'center of projection' of the artist represents the 'view point'; the surface of the picture plane represents the screen. This illustrates how one might determine what point in 3-d space corresponds to a given point on the screen.
Image courtesy "The Arrow In The Eye" by Michael Kubovy, Christopher Tyler and WebExhibits.

BY Richard Hughes and Elizabeth Power **FOR** Chris Terman **IN** 6.111 **ON** Saturday, October 25, 2008



The Phong Reflection Model

To calculate the color of a pixel once we've determined what object the ray intersects with, we sum up the contributions of three distinct lighting terms. Because we do this in hardware, we can calculate all three color contributions in parallel, then sum them together to save time. Because we

- First, we determine the effect of ambient lighting - a constant low color.

- Then, we calculate the diffuse color - a color whose brightness is directly proportional to the lighting, which in turn is a function of, for each light that is not shadowed, the distance from the surface to the light and the cosine of the angle of the surface deflection from the light - i.e, a surface directly facing the light source receives all the light, a surface perpendicular to the light source receives no light, and a light source at a 45 degree angle receives square_root (1/2) the light.

- Next, we determine the specular lighting from each non-shadowed light source, usually a bright white highlight that drops off extremely rapidly as the view-point moves away from the angle that directly reflects a source of light.

- Finally, we add them all together to determine the light for that pixel.

Superior Game Complexity

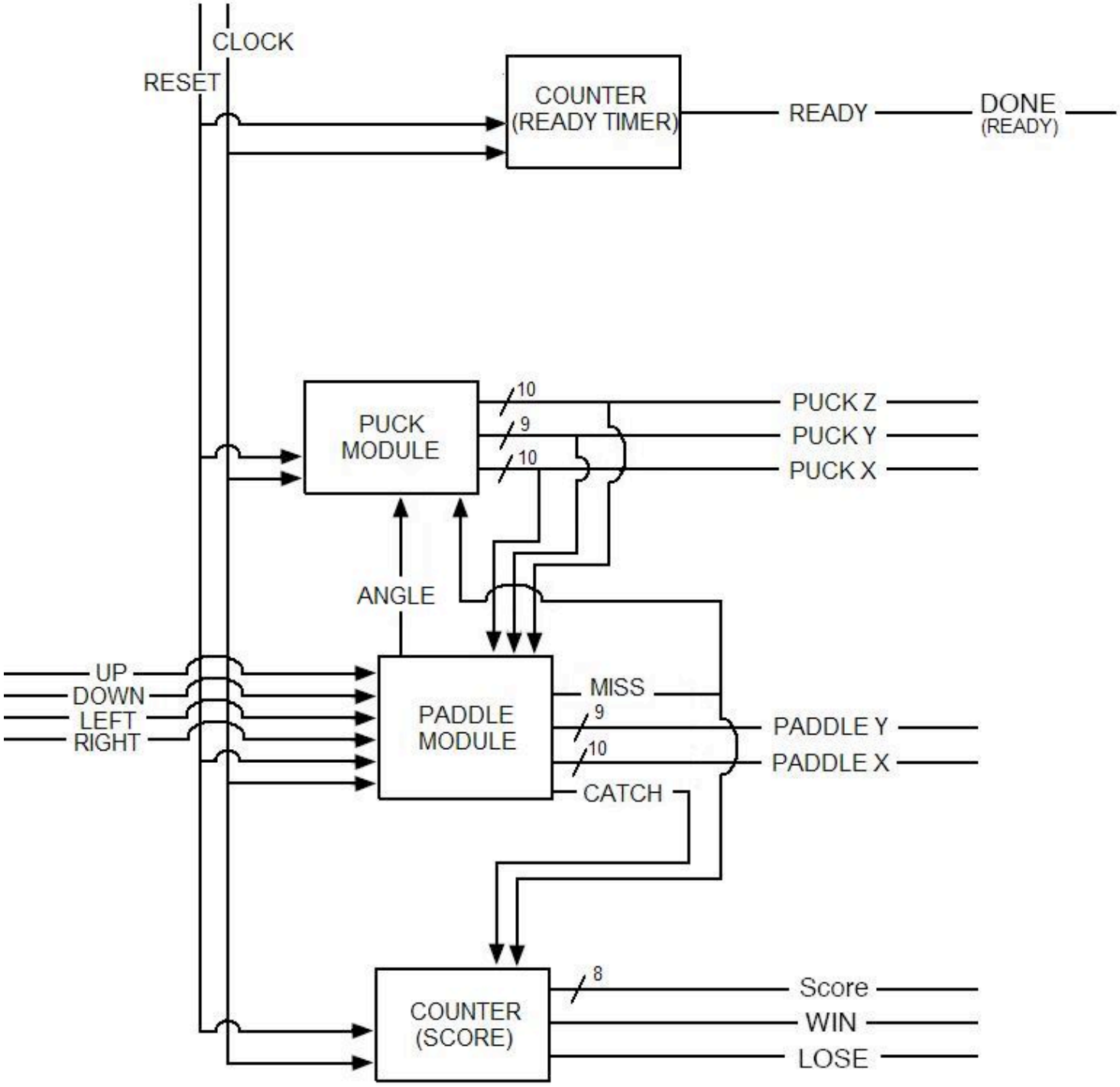
In the original pong game, the physics modeling was simple - the ball bounced at a perfect reflective angle, lost no energy from the bounce, and moved at a constant speed. Our game will be more complex.

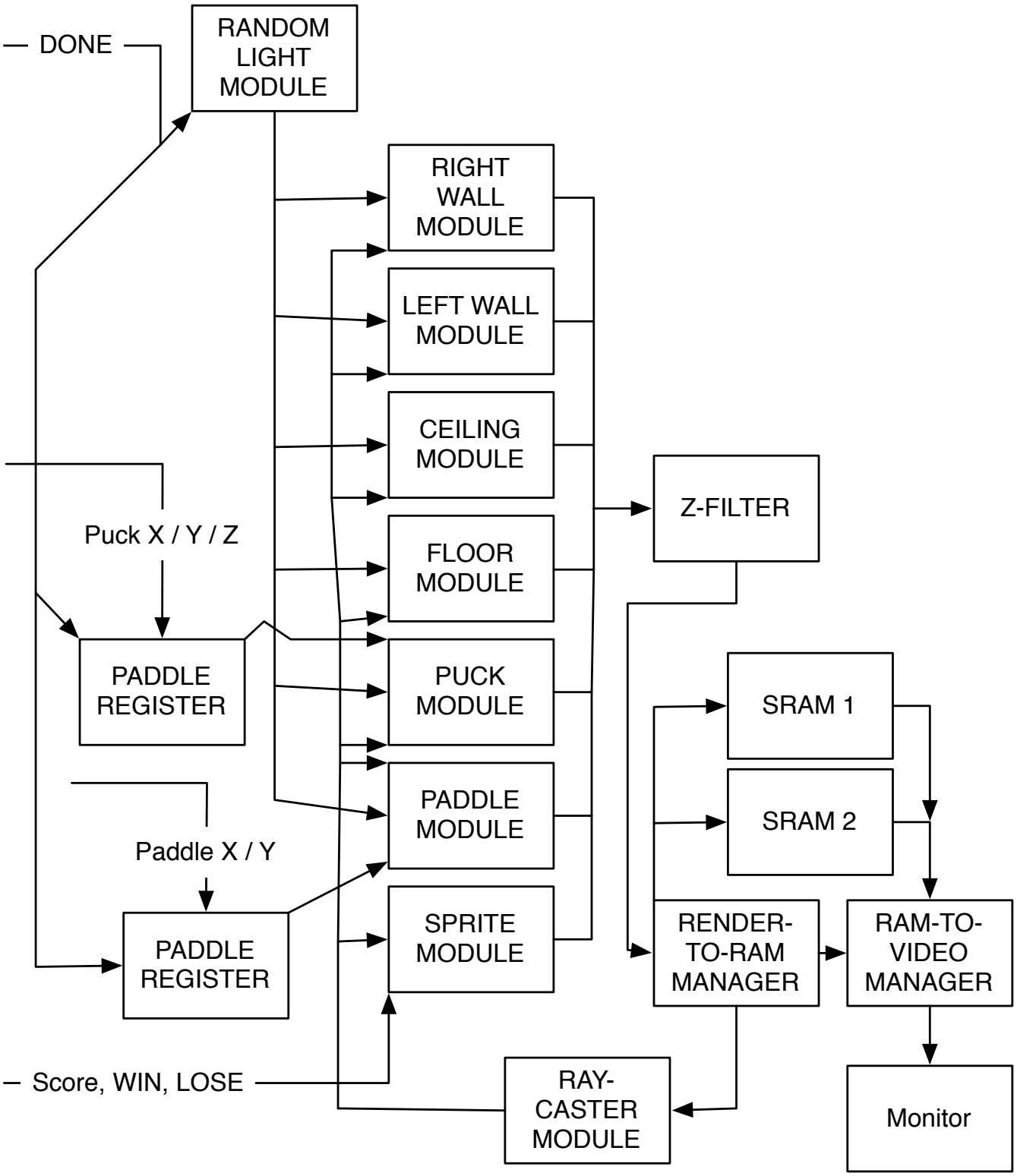
- Each time the ball bounces off the paddle, the ball will gain speed, so as to make the game more difficult.

- The paddle will have inertia, so that stopping it and starting it takes time, and it will also bounce off the walls.

- As the score approaches the maximum of 63, the ball will begin to 'wobble' in mid air, making it more difficult to intercept it with the paddle.

Appendix: Circuit Block Diagrams




DONE
RANDOM LIGHT MODULE
RIGHT WALL MODULE
LEFT WALL MODULE
CEILING MODULE
FLOOR MODULE
PUCK MODULE
PADDLE MODULE
SPRITE MODULE
Z-FILTER
Puck X / Y / Z
PADDLE REGISTER
Paddle X / Y
PADDLE REGISTER
SRAM 1
SRAM 2
RENDER-TO-RAM MANAGER
RAM-TO-VIDEO MANAGER
Monitor
Score, WIN, LOSE
RAY-CASTER MODULE