A Simple, Real Time Ray Tracer

Teammates

Daniel Benamy

Keerti Joshi

Dave Smith

Summary

Our ray tracer is going to be extremely simple. We will use many assumptions to make our job easier, but even so the results should look pretty interesting. If we have time, we'll implement a basic game. Otherwise we'll just show some cubes floating around.

Assumptions

1. Our scene is positioned on a (x, y, z) grid.
2. x, y, and z are always greater than 0 and less than 5000.
3. Our scene will consist entirely of rectangles.
4. Every rectangle will be parallel to the plane x = 0, y = 0, or z = 0.
5. A diffuse light magically appears from everywhere, so there are no shadows.
6. Light rays mysteriously reflect only three times at most.
7. Every surface contributes 50% of its own color, and 50% reflected color.
8. The camera will always look along an axis parallel to the x-axis. So x is always depth, y is side to side, and z is height.
9. Our scene will contain a limited number of rectangles, the exact number of which we will determine later.
10. Rectangles have a single color and no texture.

Prototype

This c code is a prototype of the system we will build. I did not include vision_utilities.c or vision_utilities.h (courtesy of Prof. Shree Nayar) which simply enabled this prototype to save an image. All the interesting code is in this file.

Note that we only require integers. IntersectionHelper is very important, as we call it twice per pixel per rectangle. This function is also our most difficult as it requires integer division, and multiplication. Otherwise, our prototype only requires comparison, addition, multiplication by -1, and shifting.

```
#include <stdio.h>
#include <stdlib.h>
#include "vision_utilities.c"

/*** The Scene ***/

const int RECTANGLES = 9;
const int INFINITY = 5000;
const int ROWS = 480, COLS = 640;
const int CAMERA_X_SLOPE = 200;
const int CAMERA_X = 0, CAMERA_Y = 512, CAMERA_Z = 100;

// Rec[0] is the floor, Rec[1] is the back, Rec[2] is the ceiling.
// Rec[3 - 5] are the left pseudo cube. Rec[6 - 8] are the right cube.
int recCorner1X[] = {0, 1023, 0, 500, 500, 500, 500, 500, 500};
int recCorner1Y[] = {0, 0, 0, 700, 700, 700, 300, 300, 300};
int recCorner1Z[] = {0, 0, 1023, 200, 200, 200, 200, 200, 200};

int recCorner2X[] = {1023, 1023, 1023, 500, 600, 600, 500, 600, 600};
int recCorner2Y[] = {1023, 1023, 1023, 800, 800, 700, 200, 200, 300};
int recCorner2Z[] = {0, 1023, 1023, 300, 200, 300, 300, 200, 300};

int recRed[] = {255, 255, 0, 0, 255, 0, 255, 0, 255};
int recGreen[] = {255, 0, 255, 255, 255, 0, 0, 255, 255};
int recBlue[] = {0, 0, 0, 0, 0, 255, 255, 255, 255};

/*** Intersection ***/

int IntersectionHelper(int start, int slope, int known, int startKnown,
        int knownSlope) {
    if (knownSlope == 0 || start == INFINITY) {
        return INFINITY;
    } else {
        // This is the hard part.
        int returnValue = start + slope * (known - startKnown) / knownSlope;
        // The ray must move forward.
        if ((slope <= 0 && returnValue <= start) ||
                (slope >= 0 && returnValue >= start)) {
            return returnValue;
        } else {
            return INFINITY;
        }
    }
}

int red, green, blue; // color of intersection
int intersectX, intersectY, intersectZ; // point of intersection
int reflectX, reflectY, reflectZ; // direction of reflection
void SetIntersection(xStart, yStart, zStart, xSlope, ySlope, zSlope) {
    red = 0; green = 0; blue = 0; // Background color
    intersectX = INFINITY; intersectY = INFINITY; intersectZ = INFINITY;
    reflectX = xSlope; reflectY = ySlope; reflectZ = zSlope;
    int closestDistance = INFINITY, nextDistance;
    int x, y, z; // where the ray intersects the plane for this rectangle
    int rec;
```

```
    for (rec = 0; rec < RECTANGLES; rec ++) {
        if (recCorner1X[rec] == recCorner2X[rec]) {
            x = recCorner1X[rec];
            y = IntersectionHelper(yStart, ySlope, x, xStart, xSlope);
            z = IntersectionHelper(zStart, zSlope, x, xStart, xSlope);
        } else if (recCorner1Y[rec] == recCorner2Y[rec]) {
            y = recCorner1Y[rec];
            x = IntersectionHelper(xStart, xSlope, y, yStart, ySlope);
            z = IntersectionHelper(zStart, zSlope, y, yStart, ySlope);
        } else if (recCorner1Z[rec] == recCorner2Z[rec]) {
            z = recCorner1Z[rec];
            x = IntersectionHelper(xStart, xSlope, z, zStart, zSlope);
            y = IntersectionHelper(yStart, ySlope, z, zStart, zSlope);
        } else {
            printf("Rec %d not parallel to x = 0, y = 0, or z = 0", rec);
            exit(1);
        }
        // Is the intersection inside our coordinate system?
        if (x >= 0 && x < INFINITY && y >= 0 && y < INFINITY && z >= 0
                && z < INFINITY) {
            // Is the intersection actually inside the rectangle?
            if (
                    (
                        (x <= recCorner1X[rec] && x >= recCorner2X[rec])||
                        (x >= recCorner1X[rec] && x <= recCorner2X[rec])
                    ) && (
                        (y <= recCorner1Y[rec] && y >= recCorner2Y[rec])||
                        (y >= recCorner1Y[rec] && y <= recCorner2Y[rec])
                    ) && (
                        (z <= recCorner1Z[rec] && z >= recCorner2Z[rec])||
                        (z >= recCorner1Z[rec] && z <= recCorner2Z[rec])
                    )
                ) {
                // Is the intersection closer than the previous closest?
                if (x < xStart) {
                    nextDistance = xStart - x;
                } else {
                    nextDistance = x - xStart;
                }
                if (nextDistance > 0 && nextDistance < closestDistance) {
                    closestDistance = nextDistance;
                    red = recRed[rec]; green = recGreen[rec];
                    blue = recBlue[rec];
                    intersectX = x; intersectY = y; intersectZ = z;
                    reflectX = xSlope; reflectY = ySlope;
                    reflectZ = zSlope;
                    if (recCorner1X[rec] == recCorner2X[rec]) {
                        reflectX *= -1;
                    } else if (recCorner1Y[rec] == recCorner2Y[rec]) {
                        reflectY *= -1;
                    } else if (recCorner1Z[rec] == recCorner2Z[rec]) {
                        reflectZ *= -1;
                    }
                }
            }
```

```
        }
    }
}

/*** Run the ray trace ***/

int main(int argc, char **argv) {
    ImageColor newImg;
    int col, row;
    setSizeColor(&newImg, ROWS, COLS);
    setColorsColor(&newImg, 255);

    int imageRow, imageColumn;
    int currentRed, currentGreen, currentBlue;
    for (col = -COLS / 2; col < COLS / 2; col ++) {
        for (row = -100; row < ROWS - 100; row ++) {
            SetIntersection(CAMERA_X, CAMERA_Y, CAMERA_Z,
                CAMERA_X_SLOPE, col, row);
            currentRed = red / 2; currentGreen = green / 2;
            currentBlue = blue / 2;
            // 3 reflections
            SetIntersection(intersectX, intersectY, intersectZ, reflectX,
                reflectY, reflectZ);
            currentRed += red / 4; currentGreen += green / 4;
            currentBlue += blue / 4;
            SetIntersection(intersectX, intersectY, intersectZ, reflectX,
                reflectY, reflectZ);
            currentRed += red / 8; currentGreen += green / 8;
            currentBlue += blue / 8;
            SetIntersection(intersectX, intersectY, intersectZ, reflectX,
                reflectY, reflectZ);
            currentRed += red / 16; currentGreen += green / 16;
            currentBlue += blue / 16;

            imageRow = ROWS - (row + 100);
            imageColumn = COLS - (col + COLS / 2);
            setPixelColor(&newImg, imageRow, imageColumn, currentRed,
                currentGreen, currentBlue);
        }
    }
    writeImageColor(&newImg, "RayTrace.pmg");
    return 0;
}
```

Example

This image is the output of the above prototype.