CS241 Project 5

Synopsis

Produce a complex scene using the transformation class and all the transformation, clipping, and scan-conversion code developed this semester. Add Phong lighting and Gouraud shading to your code.

Modeling Instructions

This project builds on the previous project's code in **render.cpp,** but in this case you replace the contents of **scene.cpp** with code to create your own scene. Use the transformation class, in particular its **stack** of transformations and the **Push** and **Pop** methods to model a scene.

- An example **Cart** scene is provided. You may use this as a starting point, but be sure to create your own scene.
- Create a scene of complexity at least as great as the example scene. That is:
  - Create several base objects, each in their own coordinate system.
  - Use at least three levels of modeling hierarchy.
  - Instance calls follow this (or an equivalent) pattern:
    Push()
    Various Translates, Rotates, or Scales
    Call DrawSomething(scene)
    Pop()
- You must demonstrate an effective use of modeling transformations.
- Be creative, have fun, and ultimately, understand the power of hierarchical modeling allowed by this implementation of 3D transformations.

Phong Lighting Instructions

Calculate Phong lighting at each pixel and display the computed color. Use Gouraud shading (that is interpolate normals across each triangle). The values used in

$$I = I_a K_d + \sum_i I_i \left(K_d (N \cdot L_i) + K_s (V \cdot R_i)^n \right)$$

can be found in various places in the framework:

- Scene/Light values:
  - I_a (ambient light): scene.ambient
  - Number of lights: scene.lights.size()
  - I_i (color of i^{th} light): scene.lights[i].color
- Material values:
  - K_d (diffuse reflectivity): Object::Kd
  - K_s (diffuse reflectivity): Object::Ks
  - n (specular exponent): Object::n
- Geometry values (Don't forget to make each a unit length vector.):
  - N (surface normal): Originally Vertex::N for each vertex, then interpolated through the clipper and scan conversion to a normal at each pixel (just like texture coordinates). Note: This normal interpolation is called Gouraud shading.
  - L_i (direction of i^{th} light): scene.lights[i].position - pixel in world coordinates
  - V (direction of Eye): Eye in world coordinates - pixel in world coordinates
  - R_i (reflection direction): $R_i = 2(L_i \cdot N)N - L_i$

  where:
  - Eye (in world coords): scene.viewing.BackTransform of (0,0,0,1)
  - Pixel (in world coords): Start with pixel (x,y,z) from the scan conversion, back transform via scene.projection.BackTransform, and then via scene.viewing.BackTransform, and then perform a homogeneous division.

BackTransform code

In order to perform the back transformations, the following code should be added to **transformation.cpp**, and it's calling signature to **transformation.h.**

```
Vector4D Transformation::BackTransform(const Vector4D& V)
{
        return Vector4D(
         inv[0][0]*V[0] + inv[0][1]*V[1] + inv[0][2]*V[2] + inv[0][3]*V[3],
         inv[1][0]*V[0] + inv[1][1]*V[1] + inv[1][2]*V[2] + inv[1][3]*V[3],
         inv[2][0]*V[0] + inv[2][1]*V[1] + inv[2][2]*V[2] + inv[2][3]*V[3],
         inv[3][0]*V[0] + inv[3][1]*V[1] + inv[3][2]*V[2] + inv[3][3]*V[3]
        );
}
```

Project Report

Please submit a **succinct** report with your project. Place it in a text file named **report.txt** or **report.doc** or **report.odt** if you (unnecessarily) insist on producing a formated document.

To submit

Use Moodle to submit your project via the “CS241 Project 5” link on the class web page. Be sure to submit a **zip** of any source files that you have changed since starting this sequence of projects. If you have added any files of you own, be sure to also include the *.vcproj file.

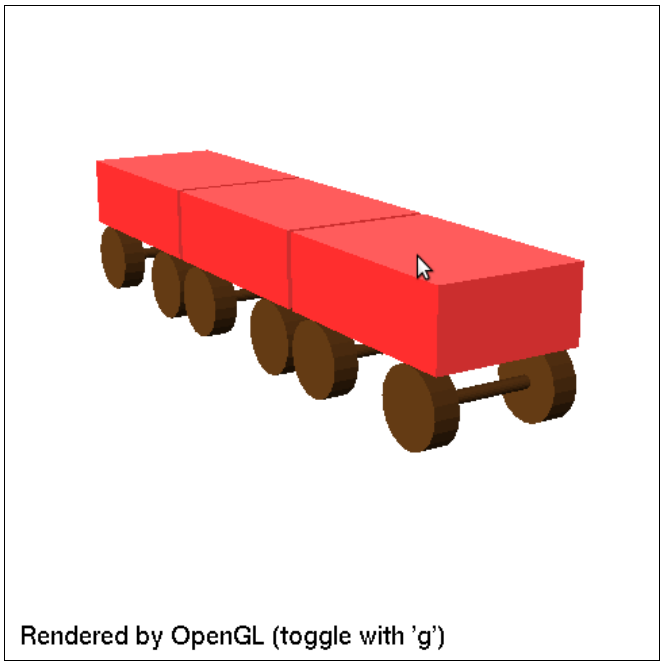
Grading Basis

Grading will be on a 100 point basis, with the following distribution:

- **Accuracy**: 70%, judgment based on looking at the results on the screen, **AND** examining the code to verify that you have effectively used the modeling transformation pattern.
- **Code**: 20%, judgment based on looking at the code for reasonable coding and commenting practices.
- **Project report:** 10%, judgment based on how succinctly and accurately the project report describes your implementation.

Example hierarchical scene and resulting image

```
// Draw a axle and wheels as 12 inch diameter cylinders, 20 units apart.
void DrawAxle(Scene& scene)
{
   scene.SetColor(HSVColor(0.08, 0.8, 0.4), HSVColor(0.0,0.0,0.2), 110);

          CreateCylinder(scene, 1.0, 20.0, 20);

          scene.modeling.Push();
          scene.modeling.Translate(0, 0, 10);
          CreateCylinder(scene, 6.0, 4.0, 20);
          scene.modeling.Pop();

          scene.modeling.Push();
          scene.modeling.Translate(0, 0, -10);
          CreateCylinder(scene, 6.0, 4.0, 20);
          scene.modeling.Pop();
}

// Draw a box centered at the origin of size 3 by 1 by 2 (feet)
void DrawBody(Scene& scene)
{
   scene.SetColor(HSVColor(0.0, 1.0, 1.0), HSVColor(0.0,0.0,0.2), 0);
   CreateRectangularPrism(scene, 3, 1, 2);
}

// Draw a cart from a body and two wheels.
//  Extends from 0 to 3 along the X axis
//  Sits on the x axis and extends from 0 to 2 in Y
void DrawCart(Scene& scene)
{
     scene.modeling.Push();
     scene.modeling.Translate(1.5, 1.5, 0.0);
     DrawBody(scene);
     scene.modeling.Pop();

     scene.modeling.Push();
     scene.modeling.Translate(0.5,0.5, 0);     // 2nd: Translate to left wheel
     scene.modeling.Scale(1/12.0, 1/12.0, 1/12.0);  // 1st: Scale to feet
     DrawAxle(scene);
     scene.modeling.Pop();

     scene.modeling.Push();
     scene.modeling.Translate(2.5,0.5, 0);     // 2nd: Translate to right wheel
     scene.modeling.Scale(1/12.0, 1/12.0, 1/12.0);  // 1st: Scale to feet
     DrawAxle(scene);
     scene.modeling.Pop();
}


// Draw a train of carts.
//   All three sit on the X axis,
//   are 3 units long, and
//   are separated by 0.1 units (in feet),
// for a total of 0 to 9.2 in X and 0 to 2 in Y
void DrawTrain(Scene& scene)
{
   DrawCart(scene);                // First cart -- no transformations needed.

   scene.modeling.Push();
   scene.modeling.Translate(3.1,0,0);
   DrawCart(scene);                // Second cart 3.1 units to the right
   scene.modeling.Pop();

   scene.modeling.Push();
   scene.modeling.Translate(6.2,0,0);
   DrawCart(scene);                // Third cart 2*3.1 units to the right
   scene.modeling.Pop();
}

void DrawWorld(Scene& scene)
{
   scene.modeling.Identity();
   scene.modeling.Translate(0, -0.7, 0); // 3rd: Translate down to a ground
   scene.modeling.Scale(0.2, 0.2, 0.2); // 2nd: Scale train to [-1,1] in X
   scene.modeling.Translate(-4.6, 0,0); // 1st: Translate train center to origin

   scene.modeling.RotateX(90);
   DrawTrain(scene);
}

void CreateScene(Scene& scene)
{
   scene.Clear();
   glEnable(GL_NORMALIZE);

   scene.EnableLighting = true;
   scene.EnableFlatShading = true;

   scene.SetAmbient(Color(0.25, 0.25, 0.25));
   scene.AddLight(Point3D(10,20,10), Color(0.9, 0.9, 0.9));
   scene.AddLight(Point3D(-10,-20,1), Color(0.9, 0.9, 0.9));

   DrawWorld(scene);
}
```



Rendered by OpenGL (toggle with 'g')