Middle East Technical University | Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2020-2021
Assignment 2 - Textures and Transformations

Due date: December 13th, 2020, Sunday, 23:59

1 Objectives

In this assignment, you are going to extend the basic ray tracer in Assignment 1 with basic texture mapping and transformation capabilities by using the techniques you have learned in the class. You must use C/C++ in your implementations (or ask permission from the instructors for other options). A solution for Assignment 1 is provided as homework files. You may start Assignment 2 from this solution or use your own ray tracer from HW1, which is the recommended option.

Keywords: *ray tracing, texture mapping, modeling transformations*

2 Specifications

1. Every specification from Assignment 1 also holds in this assignment unless it is changed and mentioned below.
2. **You must provide a makefile to compile your program.** This file must use g++ as the compiler. You are free to choose your compile options. Failing to provide a working makefile will have a penalty of 5 points out of 100.
3. The scene will be composed of triangles, meshes, and spheres. These are the same object types as in HW1.
4. You are going to implement texture mapping. The XML file will have the necessary tags to support this feature. Texture images will be given in JPEG format, for which a reader will be provided.

3 The Scene File

The scene file will be formatted as an XML file with the same format as in HW1. This section describes only the different elements that are required for the current homework. **The IDs for each type of element will start from 1** and increase sequentially. Also, notice that every number represented by X, Y, and Z are floating point, whereas every number represented by N are integers and S denotes a string.

1. **Transformations:** Scaling, rotation and translation will be defined separately and there can be multiple transformations of each type.

   - Scaling is defined by three values representing scaling values for each axis.
   - Translation is defined by three values representing translation values for each axis.
   - Rotation is represented by an angle (in degrees) and a vector around which the rotation will be performed. Positive angles indicate counter-clockwise direction.

2. **Texture:** Each texture will have "ImageName", "Interpolation", "DecalMode" and "Appearance" tags.

   - **"ImageName"** represents the name of the image file.
   - **"Interpolation"** represents the type of interpolation technique used during texture sampling. It can be either "nearest" which stands for nearest-neighbor or "bilinear" which stands for bilinear interpolation.
   - **"DecalMode"** represents how you should use the R, G, B values that you acquired from the texture. Suppose that you have computed the uv texture coordinates for the point that the ray has intersected. You will use these coordinates to fetch the necessary color information from the texture. How you should use this color information depends on "DecalMode".

     Recall that we calculate the diffuse component for shading with the following formula:

     $$I_d = k_d \; I_{incident} \; cos\theta$$

     If "DecalMode" is "replace_kd", the formula will be:

     $$I_d = C \; I_{incident} \; cos\theta$$

     If "DecalMode" is "blend_kd", the formula will be:

     $$I_d = \left(\frac{k_d + C}{2}\right) \; I_{incident} \; cos\theta$$

     If "DecalMode" is "replace_all", the formula will be:

     $$I_d = C$$

     Note that in "replace_kd" and "blend_kd" modes $C = \frac{TextureColor}{255}$ and in "replace_all" $C = TextureColor$.

- **"Appearance"** represents how you should interpret the uv coordinates that are outside the range [0,1]. "Clamp" indicates that any uv value greater than 1 should be set to 1 and any value less than 0 should be set to 0, while the values between 0 and 1 should stay the same. "Repeat" indicates that $uv_{final} = uv_{initial} - \lfloor uv_{final} \rfloor$

3. **TexCoordData:** Each line contains a texture coordinate that belongs to the vertex defined in the same line of VertexData.

4. **Object-Transformations:** Each object might have a "Transformations" tag. It contains series of transformations which should be applied to the object in the given order. For example, s_1 s_2 r_6 t_4 s_5 t_6 should first apply the scaling with id 1. Then scaling with id 2, rotation with id 6, translation with id 4 and so on.

5. **Object-Texture:** Each object can have a texture tag which represents the id of the texture to be used for this object.

4 Hints & Tips

1. First refer to the HW1 instructions if something is not clear. You may find the answer there

2. In general, you don't need to compute the inverse of a matrix through algebraic methods. All you need to do is apply reverse transformations in reverse order to get the inverse of a composite transformation matrix. In any case, a sample source code will be provided that shows how you can invert a 4x4 matrix.

3. Rotating a sphere around its center has a visible effect only if a texture is mapped on it. To rotate a texture mapped sphere, define a local uvw coordinate system for the sphere, rotate this coordinate system, and compute the spherical coordinates with respect to this rotated system.

4. Remember that normal vectors must be transformed with the inverse-traspose of the transformation matrix (upper 3x3 part). Also make sure to renormalize your normal vectors after transformations.

5. Spheres will not be transformed by non-uniform scaling. That is after the transformations, a sphere will remain as a sphere (as opposed to turning into an ellipsoid).

5 Bonus

There is no speed bonus this time. But with texture mapping and transformations, you can now create quite realistic scenes and images. If you make interesting scene contributions you can get a 5 point bonus.

6 Regulations

1. **Programming Language:** C/C++. Ask your instructor if you want use another language.

2. **You are not allowed to use external matrix libraries.** The typical matrix operations you will perform are matrix multiplication, transpose, and inverse; all of which should be simple to implement. A sample inversion routine will also be provided in *matrixInverse.cpp* file, you can use that to invert 4x4 matrices.

3. **Late Submission:** Refer to the syllabus for the late policy.

4. **Groups:** You can team-up with another student. But you must notify the assistants about who your partner is within the allowed time frame.

5. **Cheating:** We have zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations and will get 0 from the homework. You can discuss algorithmic choices, but sharing code between groups or using third party code is strictly forbidden. To prevent cheating in this homework, we also compare your codes with online ray tracers and previous years' student solutions. In case a match is found, this will also be considered as cheating. Even if you take only a "part" of the code from somewhere or somebody else, this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar.

6. **Newsgroup:** You must follow the ODTUCLASS forum for discussions, possible updates, and corrections.

7. **Submission:** Submission will be done via ODTUCLASS. Create a "tar.gz" file that contains all your source code files and a makefile. The executable should be named as "raytracer" and should be able to be run using the command "./raytracer scene_file_name.xml". **The .tar.gz file name should be:**

   - If a student works with a partner student:

     `<partner_1_student_id>_<partner_2_student_id>_rasterizer.tar.gz`

   - If a student works alone:

     `<student_id>_rasterizer.tar.gz`

   - For example:

     ```
     1234567_2345678_rasterizer.tar.gz
     1234567_rasterizer.tar.gz
     ```

8. **Evaluation:** Your codes will be evaluated on Inek Machines; based on several input files including, but not limited to the test cases given to you as examples. Rendering all scenes correctly within the time limit (30 mins max per scene) will get you 100 points.

7 Sample Scene File

```
<Scene>
        <!-- Only new elements are shown in this example -->
        <Transformations>
                <Scaling id="ScaleId">X Y Z</ Scaling>
                <Rotation id="RotationId">Angle X Y Z</ Rotation>
                <Translation id="TranslationId">X Y Z</ Translation>
        ...
        </Transformations>

        <Textures>
                <Texture id="TextureId">
                        <ImageName>S</ ImageName >
                        <Interpolation>{nearest, bilinear}</ Interpolation>
                        <DecalMode>{replace_kd, blend_kd, replace_all}</ DecalMode>
                        <Appearance>{repeat, clamp}</ Appearance >
                </Texture>
                ...
        </Textures>

        <TexCoordData>
                T1u T1v
                T2u T2v
                T3u T3v
                ...
        </TexCoordData>

        <Objects>
                <Mesh id="MeshId">
                        <Transformations>{s1 r2 t1 ...}</ Transformations>
                        <Material>N</Material>
                        <Texture>N</Texture>
                        ...
                </Mesh>
                <Triangle id="TriangleId">
                        <Transformations>{r3 t2 s1 s2 ...}</ Transformations>
                        <Material>N</Material>
                        <Texture>N</Texture>
                        ...
                </Triangle>
                <Sphere id="SphereId">
                        <Transformations>{t1 t2 r2 r1 s2 ...}</ Transformations>
                        <Material>N</Material>
                        <Texture>N</Texture>
                        ...
                </Sphere>
        </Objects>
</Scene>
```