Middle East Technical University | Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2016-2017
Assignment 2 - Texture Mapping and Transformations in Ray Tracing

Due date: 21 November 2016, Monday, 23:59

1 Objectives

In this assignment, you are going to extend the basic ray tracer in Assignment #1 with texture mapping and transformation capabilities by using the techniques you have learned in the class. You must use C/C++ in your implementations. The solution to Assignment #1 is provided in COW. You may start with this solution or use your own ray tracer from HW1.

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

2 Specifications

1. You should name your executable as "raytracer".
2. Your executable will take two txt files as argument: one that describes the virtual environment and one that describes the cameras as argument (e.g. "sample_scene.txt sample_camera.txt"). The format of the camera file is the same as in Assignment #1. The format of the scene file is described in the next section. You should be able to run your executable via the command "./raytracer sample_scene.txt sample_camera.txt".
3. As in the first assignment, the camera description file may contain multiple camera configurations. You should render as many images as the count of cameras. The output filenames for each camera is also specified in the camera input file.
4. Similar to the first assignment, you will save the resulting images as PPM files.
5. You will have at most 30 minutes to render scenes for each input file on inek machines. Programs exceeding this limit will be killed and will be assumed to have produced no image. Therefore, if your program cannot produce any output at most 30 minutes, your program will get a 0 from that test case.

6. You will implement two types of light sources: point and ambient. There may be multiple point light sources and a single ambient light. The intensity values of these lights will be given as (R, G, B) triplets. You should apply attenuation as in HW1. When writing the final color, if you find a color value greater than 255 you must clamp that value to 255 (also you must never find negative values).

7. The scene will only be composed of instances of spheres and cubes. Each object in the scene will be either an instance of a unit sphere centered at origin or an instance of a unit cube again centered at origin. However, with a sequence of several transformations (translation, rotation, scaling) you will be able to generate multiple instances of these objects with different appearances at different positions, orientations, and size. Transformations will be applied to the object instances in the order specified in the input file.

8. You will use texture maps to replace the incident light for the diffuse color component. In HW1, you were supposed to compute the diffuse color as follows:

$$I_d = k_d I_{attenuated} cos(\theta) \tag{1}$$

Now, instead, you will compute it as:

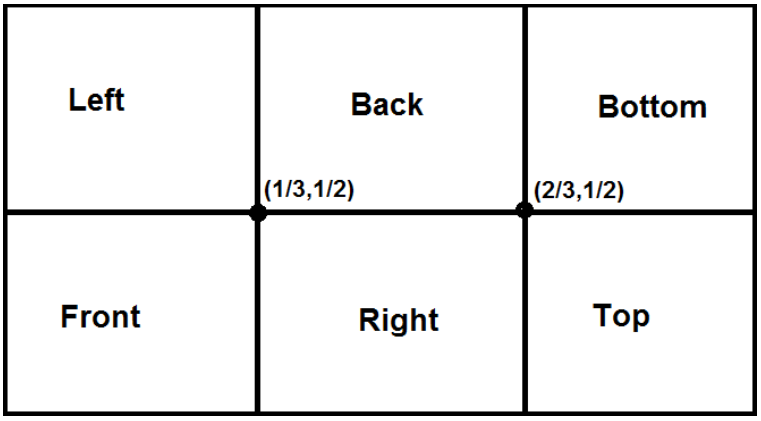
$$I_d = k_d C cos(\theta) \tag{2}$$

where C represents the texture color. The texture color values will be in $[0, 255]$ range. You are expected to use *nearest-neighbor* interpolation method when fetching the texture color from the texture image.

9. Texture images will be given in JPEG format. A sample JPEG reader will be provided to you with the homework. $uv = (0, 0)$ indicates the top-left corner of the texture and $uv = (1, 1)$ represents its bottom-right corner.

10. Since a sphere may no longer be a sphere after non-uniform scaling, we will only scale spheres by uniform scaling.

11. The texture to be mapped on a cube will be given as a single rectangular texture and different parts of the texture will be mapped to the cube's faces as shown in the figure below (numbers shown are uv coordinates).



The figure shows how the texture should be mapped to the cube in its untransformed state, i.e., centered at the origin and front face facing the the positive z-axis.

3 Scene File

A sample scene file looks like the following:

- **Ray Reflection Count** Num
- **Background Color** R G B
- **Ambient Light** R G B
- **Point Light Count** Num
    - **Position** X Y Z
    - **Intensity** R G B
- **Material Count** Num
- **Material "Mid"** id
    - **Ambient** R G B
    - **Diffuse** R G B
    - **Specular** R G B SpecExp
    - **Reflectance** R G B
- **Textures**
- **Texture Count** Num
    - **Texture file name** String
- **Translation**
- **Translation Count** Num
    - t_x t_y t_z
- **Scaling**
- **Scaling Count** Num
    - s_x s_y s_z
- **Rotation**
- **Rotation Count** Num
    - α u_x u_y u_z
- **Instance Count** Num
- **Instance** CubeInstance or SphereInstance
    - **Material Id** Mid
    - **Texture Id** Tid
    - **Transformation Count** Num
        - **Transformation**$_{type}$ **Transformation**$_{id}$

1. Ray Reflection Count
Specifies how many bounces the ray makes off of mirror-like objects. Applicable only when a material has non zero reflectance value.

2. Background Color
Specifies the R, G, B values of the background. If a ray through a pixel does not hit any object, the pixel will be of this color.

3. Ambient Light
R, G, B intensity parameters of the ambient light as floats. This is the amount of light received by each object even when it is in shadow.

4. Point Light Count
Specifies how many point lights are in the scene file. Position X, Y, Z and Intensity R, G, B values for the light source. All values are floats. The numbers of Position and Intensity lines are equal to the number of point lights.

5. Material Count
Specifies how many materials are in the scene file.

6. Material
Material with id Mid is defined with ambient, diffuse, specular, and reflection properties for each color channel. Values are floats between 0.0 and 1.0. SpecExp defines the specularity exponent in Phong shading. Reflectance represent the degree of the mirrorness of the material. If this value is not zero, you must cast new rays and scale the resulting color value with the specified parameters.

7. Texture Count
Specifies how many texture images will be loaded.

8. Texture file name
Specifies the file name of the texture image.

9. Translation Count
Specifies how many translations are defined in the scene file.

10. Translation parameters
t_x, t_y, and t_z are the translation parameters, i.e., translation amounts along the major axes.

11. Scaling Count
Specifies how many scale transformations are defined in the scene file.

12. Scaling parameters
s_x, s_y, and s_z are the scaling parameters, i.e., scaling level in the corresponding coordinate axes.

13. Rotation Count
Specifies how many rotations are defined in the scene file.

14. Rotation parameters
α, u_x, u_y, and u_z are the rotation parameters, i.e., the object is rotated α degrees around the rotation axis which pass through points (u_x, u_y, u_z) and $(0, 0, 0)$. The positive angle of rotation is given as the counter-clockwise rotation along the direction (u_x, u_y, u_z).

15. Number of Instances
Specifies how many cube and sphere instances are in the scene file in total.

16. Object instance

1. Material ID of the instance is Mid.
2. Texture ID of the instance is Tid. Texture ids are in the range [0..Number of Textures]. If the texture id is 0, it means that no texture will be applied on this instance and the diffuse reflectance coefficients defined in the material description will be used for this instance. If texture id is greater than 0, it means that the corresponding texture will be applied on this instance by replacing the material diffuse reflectance coefficient with the texture color.
3. The sequence of transformations listed are applied on the vertices of cubes and center of the sphere in their order of appearance. Scale transformation for a sphere instance is applied on the sphere radius only.
4. The transformation type 't' indicates a translation transformation, 's' indicates a scale transformation, and 'r' indicates a rotation transformation. The transformation id ranges from [1..Number of Translations] for type 't' transformations, and similarly for the other type of transformations.

4 Hints & Tips

1. Start early!
2. Make sure your program compiles using the make command. A sample Makefile is given to you. You can modify this file if necessary. You are free to choose your compile options. Failing to provide a working makefile will have a penalty of 10 points out of 100.
3. You may use -O2 or -O3 option while compiling your code for optimization.
4. We will not create strange configurations such as the camera being inside a sphere, a point light being inside a sphere, or with objects in front of the image plane. If you doubt whether a special case will be tested please ask this in the newsgroup.

5. For debugging purposes, consider using low resolution images. Also it may be necessary to debug your code by tracing what happens for a single pixel (always simplify the problem when debugging).

6. If you see generally correct but noisy results (black dots), it is most likely that you are falling victim to a floating point precision error (you may be checking for exact equality of two FP numbers instead of checking if they are within a small epsilon).

7. We will evaluate your using a script first and visually if necessary. Minor differences between the expected images and your results may be tolerable.

8. Rotating a sphere with its texture may initially seem tricky. But there is a simple solution. You need to associate local up and right vectors with the sphere. Apply the given transformations to these vectors and compute θ and ϕ angles from these vectors (instead of using global y and x directions).

5 Regulations

1. **Programming Language:** C/C++

2. **Late Submission:** You can submit your codes up to 3 days late. Each late day will incur a penalty of 10 points. After 3 days, you will get 0.

3. **Cheating: We have zero tolerance policy for cheating**. There is no teaming up for computer graphics homeworks. People involved in cheating will be punished according to the university regulations and will get 0. You can discuss algorithmic choices, but sharing code between each other 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 a "part" of the code from somewhere/somebody else - this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please don't think you can get away with by changing a code obtained from another source.

4. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

5. **Submission:** Submission will be done via COW. Create a tar.gz file named "raytracer.tar.gz" that contains all your source code files and a makefile. The executable should be named "raytracer" and should be able to be run using command "./raytracer scene_name.txt camera_name.txt". If your makefile does not work (or you do not have one) and therefore we will have to manually try to compile your code - there will be an automatic penalty of 10 points.

6. **Evaluation:** Your codes will be evaluated based on several input files including, but not limited to the test cases given to you as example. Rendering all scenes correctly within the time limit will get you 100 points.