Computer Graphics - 2015/2016

Exercise 3 - Ray Tracing

Last update 23/12/2015

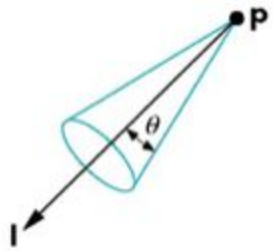
General Description

In this exercise you will implement a simple ray tracer algorithm. You will implement intersection between a ray and the following objects:

1. Convex polygon
2. Polygon mesh (using the familiar OpenMesh library)
3. Sphere
4. One additional parametric object of your choice (excluding plane).

Your ray tracer should support multisampling with a parameter (the first command line argument) that will control how many random samples will be taken through each pixel. In case this value is 1, shoot a single ray through the center of the pixel. Otherwise, shoot multiple (according to the number) random rays through the pixel and average the result.

Additionally, you will have another parameter (another command line argument) that controls the number of random samples that will be taken for reflection and refraction rays. In case this value is 1, shoot a single ray according to the perfect reflection / refraction. Otherwise, shoot multiple random rays through a cone in the direction of the reflection / refraction. A third command line argument should control the cone cutoff angle (in degrees), as in the illustration below:



By default, all available scenes (defined in SceneChooser.h) are rendered, unless a fourth command line argument is specified with a specific scene index.

In your `README.txt`, give an example command line to run your program with parameters that balance between speed and quality for each scene (i.e. try to achieve best quality but do not waste time for unnoticeable differences).

The default arguments (if only the scene index is given) are 1,1,0.

Detailed Description

0. Basics

In order to save you time and allow you concentrate on the important parts of the algorithm we provided most of the code for you. Download it from here:
http://moodle.cs.huji.ac.il/cs13/file.php/67609/ex3-template.zip

You will have to implement several functions. Before describing what you have to do, a few words about the code you get:

The class `Object` is the base class for all the objects which a ray can intersect. This class contains a virtual function intersect:

```
virtual int intersect(const Ray& ray,double& t,Point3d& P,Vector3d& N) = 0;
```

All derived classes must implement it in the following manner: It should return 1 if an intersection between ray and the object occurred, 0 otherwise. In case of intersection, the variables t, P and N are updated: t is the distance between the origin of the ray and the intersection point. P is the intersection point and N is the normal at the intersection point. Additionally, the class contains the material properties of the object (diffuse color, specular color, reflection color, etc).

The Camera class is responsible for setting the camera at the correct position and shooting the primary rays. It saves that output to a bmp file.
The Scene class contains all the objects and the lights in the scene. It is the class responsibility to check intersection between rays and objects. Also the shading computation is performed here. add_object and add_light functions are the way to add objects or lights to the scene.

Each concrete scene class is located in a separate .h file and is responsible to set the default camera parameter and the scene content including its geometry and lights.

ex5.cpp is responsible to read the arguments from the command line call a function that create the scene, and render the scene. DO NOT change this file, as we might replace your ex5.cpp with ours. To add your own scene(s), create a separate .h file for each scene, and modify SceneChooser.h accordingly (with the minimal required changes). We might edit your file with new scenes for testing your solution, so keep this (simple) file structure.

1. Complete implementation for given header files

Some header files contain unimplemented classes (e.g. `Sphere`, `Polygon` and `MyMeshObject`) you'll have to complete the missing implementation for all of these classes. You can assume the polygon is convex and you can triangulate the mesh if you want.
The class `Camera` is in charge for placing the camera at the correct location defining the image plane and shooting through the center of the pixels.

The shading model is the same as in the slides:

```
Surface_Color =

ambient term: obj.AmbientColor * scene.ambientLight +

//per each visible light source:

diffuse term: obj.diffuseColor * lightColor * <L , N> +

specular term: obj.specularColor * lightColor *<R,L>^obj.shininess

reflection term obj.reflectionColor * reflectedColor    //as returned from a
recursive trace_ray call

refracted term obj.transparecyColor * refractedColor          //as returned from
a recursive trace_ray call
```

You have to implement reflections and refraction of rays (review Snell's law). When entering a body you should take _index as the refraction index, and when leaving an object take 1/ _index. In case the refracted angle is bigger than the critical angle simply shoot a reflection ray in this direction (using material's reflection parameters).

2. Area Light Sources

The `SphereLight` class represents a spherical light source, defined by a center point, radius, and light color. Your ray tracer should support such light sources, by sending multiple shadow rays towards such a light source, to get an estimation of the fraction of light reaching the point from such light source. Describe how you implemented this feature in your `README.txt` file. Replace the first point light source in each of the given scenes with a sphere light source.

3. Custom Scenes

You'll need to create your own scene (subclass `Scene`), which will include the additional parametric object you chose to implement, an ellipsoid, and at least one area light source (`SphereLight` class), so soft shadows will appear in your scene. You can create additional scenes (extremely nice scenes will yield a small bonus). Mention it in the `README.txt` (including the executable name to generate each scene).

4. Textures (5 points)

This part is mandatory, but we make it only 5 points so you can leave (due to your time constraints) it unimplemented and still get a good grade (up to 95). If you choose not to implement this part you should set `#define WITHOUT_TEXTURES 1` in `SceneChooser.h`.

The main thing you'll have to implement is this function in object.hh (and all inheriting classes)

```
/* Returns the color of the object at point P */
/* according to the texture map and the texture coordinates assigned to this object */
/* Returns white in case of any error - no uv coordinates assigned, no texture map assigned etc.*/
virtual Color3d texture_diffuse(const Pointd & P);
```

- For sphere use angles with respect to the x and y axes as the u,v coordinates.
- For polygons find the triangle to which P belongs and use barycentric coordinates for this triangle.
- For meshes - find the triangle to which P belongs and use barycentric coordinates for this triangle.

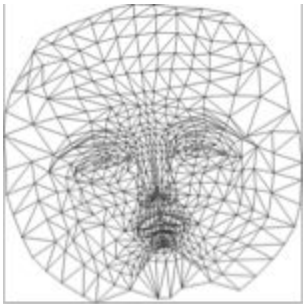
In the `shade()` function, instead of calling `diffuse()` call `diffuse(P)`. This function has been added to object.hh and it multiplies the diffuse color by the texture map.

The BImage class is used for reading the textures. to get a color value you can use the () operator: `image(x,y).r, image(x,y).r, image(x,y).b`. See example code (ex5a.cc ex5b.cc) to see how to load images. If the uv coordinates are are bigger than 1 or smaller than zero clamp them to [0,1]. Use floating point values for x,y (not integers). This way you'll get bilinear sampling of the image.

4.1. OpenMesh and texture mapping

- UV coordinates are stored in a class called `MyMesh::TexCoord2D`
- To get this class you'll have to call: `mesh.texcoord2D(vhandle);// vhandle is a VertexHandle`
- `mesh.has_vertex_texcoords2D()` is used to check if the mesh has uv coordinate stored
- `mesh.request_vertex_texcoords2D()` adds uv coordinate support to the mesh, it does not assign values.
- If you load an object with uv coordinates, they are stored in the object.
- There may be some models with different number of uv coordinates that vertices. OpenMesh does not handle these meshes well. Do not waste your time on these meshes.
- Use girl_face.obj mesh from the stuff folder as your mesh for debugging texture mapping for meshes.

- face_uv.bmp can be used to draw a new face to girl_face.obj the model: copy the image. open it in your favorite paint program and draw over it.



5. Bonus

Add one or more of the following additional features and demonstrate their effect with a scene (attach bmp files of the same scene rendered with and without these effect).

1. Depth of focus (up to 3 points)
2. Object motion blur (up to 3 points)
3. Sub-surface scattering (up to 5 points, for really high-quality results)

You are encouraged to contact us if you have other ideas.

General guidelines:

(You might lose points for not following these guidelines)

- Make sure that you understand the effect of every character that you write in your code.
- Make sure that your code does what it's supposed to do.
- Do not unnecessarily change existing code.
- Keep your code readable and clean!
    - Avoid code duplication.
    - Comment non-trivial code regions.
    - Block together (using '{' and '}') logically related lines of code.
    - Preserve coding conventions when changing existing code.
    - Do not change existing code without a good justification.
- Keep your code efficient:
    - Points might be reduced for slow running times:
      Your grade might be multiplied by `max(1, min(0.9, (school_time*1.05 / your_time)^0.5 ))`, where school_time is the run time of the school solution and your_time is the run time of your solution, measured with scenes of our choice, on a machine of our choice with compiler of our choice.
    - Your code must be thread-safe (i.e. assume that rays are sent to multiple pixels in parallel).
    - Do **NOT** use multiple threads (or processes), because we may do this on testing

your code.
    - You are encouraged to compare running times with your friends!

- **VERY IMPORTANT**: Initialize your random number generator with a pre-defined constant, to make your results reproducible.
- Add to your README.txt a list of all web-pages URLs that you used in order to complete this exercise (**do not include official reference pages**).
- Add to your README.txt a list of all students' usernames that you discussed this exercise with.

School solution

The school solution results are attached below. No executable is provided.
Rough running-times for the school solution:

- The simplest configuration runs in about 10 seconds for each scenes (Compiled with LLVM and -O3, running on a machine with 4 cores)
- More complex configurations can ran in less than a minute.
- It really depends on the hardware and the compiler, but we of-course do not expect you to perform any hardware-specific optimization.

Submission

Note: You HAVE to submit this exercise in pairs:

- Do not submit it alone without an explicit permission.
- Do not submit from multiple user names.

Include the following in your submission:

1. A `README.txt` file (not `README`), that includes:
   - your id and login
   - your partner's id and login
   - A brief description of your implementation and the changes you made to the template code provided to you.
   - A brief description of each file of the other files that you submit.
2. All files that are required for compilation of your solution with a single 'make' command, and the shaders necessary to run it.
3. Describe any bonus that you implement and explain how to check it.

DO NOT submit any file that is not necessary for your solution (e.g. .o files, backup and temporary files, etc.).
Pack all files as a single zip file named by the following pattern: ex3_<your 9 digits id>_<your_username>_<your partner's 9 digits id>_<your partner's 9 digits>_id.zip (e.g. '`ex3_123456789_mylogin_987654321_myfriendlogin.zip`').

Deadline:
You have to submit your solution (via the course's moodle webpage) no later than Monday 18/1

at 23:59.
Late submission will result in 2^(N+1) points deduction where N is the number of days between the deadline and your submission (rounded up, the minimum grade is 0, friday and saturday are excluded).