UNIVERSITÄT TÜBINGEN
Prof. Dr.-Ing. Hendrik P.A. Lensch
Lehrstuhl Computergrafik
Dennis Bukenberger (dennis.bukenberger@uni-tuebingen.de)
Zabiha Kehribar (zabiha.kehribar@student.uni-tuebingen.de)

23. November 2020

Graphische Datenverarbeitung
Assignment 3

Submission deadline for the exercises: 30. November 2020 6.00 am

Source Code Solutions

- Upload **only** the source code files listed in the description in Ilias.
- Upload them one by one and **don't** zip them.

The source code must run on `cgpool120[0-7].informatik.uni-tuebingen.de` by extracting your submitted `.tar.gz` file to a certain folder and running `scons`. You can log onto this machine via ssh by using your WSI account. From outside the WSI network, you may have to use a ssh gateway, e.g. `cgcontact.informatik.uni-tuebingen.de`
Attention: Do not use `cgcontact` for working, but only for ssh-ing to the `cgpool` machines.

The framework you get for completion already compiles and runs as requested above and you only have to modify source and header files - no files have to be created.

Written Solutions

Written solutions have to be submitted digitally as one PDF file via Ilias.

3.1 Reflection Rays (written, 5 Points)

Given a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ which hits a reflective surface at $t = t_{hit}$. The surface has the geometry normal $\mathbf{n}$ at the hit point. Assume that both the ray direction $\mathbf{d}$ and the surface normal $\mathbf{n}$ are normalized.

a) Compute the ray that has been reflected (assuming a perfect mirror reflection) by the surface.
You have to submit the solution for this exercise in written form.

3.2 Front/Back-side of a triangle (written, 5 Points)

In a 3D space, a triangle has two sides: front-side and back-side. The normal vector $\mathbf{n}$ points away from the front side. Given a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ which hits a triangle and a face normal $\mathbf{n}$ of the hit point.

a) Find a way to determine if the ray hits the face from the front or from the back.
You have to submit the solution for this exercise in written form.

Framework Update

Download the new framework from the course website. It contains an improved version of the `io_mesh_ra2` blender export script (for installation instructions see the previous assignment) which also generates a `.n` file. This file has the same format as `.ra2` but contains the vertex normals instead of its coordinates.

The framework also contains a `bunny.blend` scene for testing. Just convert it to a `.ra2` / `.n` file pair and load it into the program instead of the `Test.ra2` scene. It should appear right in front of the camera.
In addition to the solutions for the previous assignment, the updated framework contains a new, interactive camera so you can fly through the scene using the `WASD` keys and turn around by moving the mouse while holding the left mouse button.
Another addition is, that the updated framework adds a ground plane to each scene and that it preinitializes some lights which can be used for rendering.

3.3 Shading with Vertex Normals (25 Points)

Rather than calculating a single *geometry normal* for a triangle, it is often useful to store a corresponding *vertex normal* for each vertex. The advantage is that if we have a hit point on a triangle, the shading normal can be smoothly interpolated between the vertex normals. If neighboring triangles share the same vertex normals, a smooth appearance can be generated over non-smoothly tessellated geometry.

a) In order to support vertex normals, extend the `Triangle` class to contain also the vertex normals and load the `.n` file.

b) For each hit point compute the barycentric coordinates.
Hint: An easy way to calculate the barycentric coordinates is by using the area ratios in the triangle (see script). If you have two edge vectors $\vec{ab}$ and $\vec{ac}$ of a triangle, you can calculate its surface area with $\frac{1}{2}|\vec{ab} \times \vec{ac}|$.

c) Interpolate between vertex normals using the barycentric coordinates.
Note: Interpolating normalized vectors will not return a normalized vector! Make sure to normalize your interpolated normal vector.

d) Implement a simple debug shader, which uses the interpolated normal to shade the hit point. The method should just return the absolute value of the normal vector (x, y, z) as color values (r, g, b).
Note: If the appearance of the scene is not smooth, you may have missassigned the barycentric coefficients and the normal vectors during interpolation.

e) Use the interpolated normal instead of the triangle normal for diffuse shading.

3.4 Point Light Sources (20 Points)

The simple surface shading we implemented so far doesn't take light sources into account. In order to increase the rendering realism we now extend our shading model. For this, use the four preinitialized `pointLights` and the interpolated normal if you solved this task already.

a) Implement a method that calculates the light intensity based on the quadratic falloff (i.e. $\frac{1}{t^2}$, where t is the distance between the light source and the surface point), as well as the direction vector from the surface point to the light source. The direction vector will be later used for shadow computations.

b) Check for visibility by intersecting a shadow ray with the scene.
Make sure that ALL attributes of the shadow ray are initialized. Also make sure that the shadow ray does not intersect directly with the shaded object because of numeric issues by adding `RAY_EPS` to its `origin` or `tmin`. Only add a light source's contribution if it is visible!

Note: Sometimes an incident ray may hit the backside of a surface. In that case turn the shading normal around to face forward.

3.5 Mirror Shader (15 Points)

In this exercise you have to implement a mirror shader. Use the ground plane of the scene bounding box as mirror (triangles with an index >= `firstMirrorTriangle`). Each ray which hits a surface with applied mirror shader has to be reflected according to the exercise 3.1.
The reflected ray starts traversal of the scene recursively from the point of reflection. Make sure that the reflection ray doesn't hit the reflective surface again instantly because of numeric issues by adding `RAY_EPS` to the ray's origin or to `Ray::tmin`. Also make sure that you initialize ALL attributes of the mirror ray.

3.6 Area Lights (30 Points)

As you have learned in the last exercise, shadows can add important visual information to an image. Until now we have only considered point lights. Point lights create *hard shadows* because a point light can not be partly occluded and is either entirely blocked or not. To render more realistic shadows we need a more advanced light source. *Area Lights* are able to produce *soft shadows*, which are more natural. For the solution of this exercise, use the `areaLight` object (refer to the code documentation of `AreaLight` for more information) and the interpolated normal if you solved this task already.

a) Calculate the area of the light source.

b) Implement area light shading: In each illumination step cast a shadow ray from the shading point to a random point on the light source surface (use the Mersenne Twister for random number generation, you find example code in `main.cpp`).
Compute the illumination similar to the point light but incorporate the area value into your calculations: The value corresponding to the color of a point light should be the radiance * area.

c) Render an image with 1000 shadow rays per pixel: Use the mean value of 1000 light calculations as done in b) to shade a pixel.