Introduction to Computer Graphics
Exercise 5 – Toon shading

Handout date: 08.11.2012
Submission deadline: 14.11.2012, 23:00

Note

Undeclared copying of code or images (either from other students or from external sources) is strictly prohibited! Any violation of this rule will lead to expulsion from the class. Late submissions are not accepted.

What to hand in

A .zip compressed file named ”Exercise*n*-Name1-Name2.zip” where *n* is the number of the current exercise sheet and the names are the full names of the submitting students. It should contain:

- A zipped folder containing a Microsoft Visual Studio solution file with all source files and project files. Do not include the build directories in the zip. In order to avoid sending build files, close your Visual Studio solution and run the file “cleanSolution.bat” located in the top directory of the framework before you submit. Unzip and rebuild your submission to make sure it compiles without errors.
- A “readme.pdf” file describing your approach to the solution (1-2 sentences per exercise) and any problems you encountered (use the same numbers and titles). This file should be written in **English**.
- Submit your solutions to the class Moodle before the submission deadline. If you work in a pair, only one person per group needs to submit the assignment, but make sure to put both names in the title of the zip file. The enrollment key is: *introGraphics*

“Toon shading”

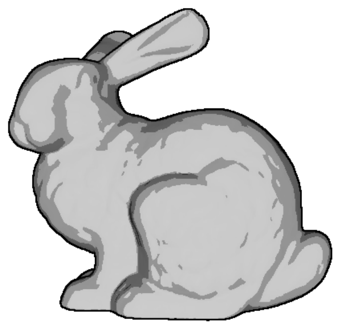


Figure 1: Toon shading of a bunny.

The goal of this exercise is to mimic the rendering style of cartoons. The non-photorealistic rendering (NPR) technique that you will implement in this practical is known as toon-shading or cel-shading. Contrary to the diffuse-shading where the colors change continuously according to the angles between the surface normals and the light, the toon-shading only uses a discrete range of tone to shade the surface. In addition to the flattened aspect given to the scene objects by this discretization it is common to draw the important contours of the scene objects in black. The combination of these two effects is shown on a bunny in Figure **??**.

"OpenGL Frame Buffer Object (FBO)"

In this practical we need to be able to render the 3D scene into a texture in order to do some post processing operations. To render the scene onto a texture we are using OpenGl Frame Buffer Objects (FBO). You will not use them explicitly in this practical because first you will implement the shaders and second the code is wrapped in the `fbo` class. Nevertheless, FBOs are really useful and most of you will use them in their project, thus it can be interesting for you to look at the `fbo` class and `CartoonViewer` class.

4.1 Introduction

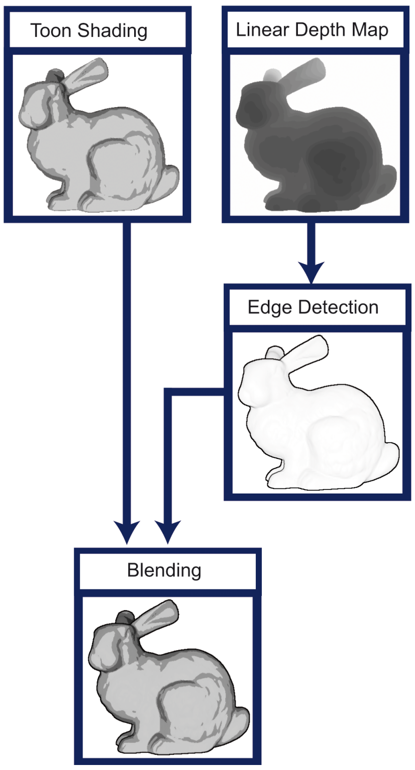


Figure 2: The cartoon rendering pipeline.

Take a look at the framework provided. The `CartoonViewer` class loads the 3D model, creates a 1D floating point texture of 12 pixels (`float tex[12]`) and loads four shaders : cartoon, depth, edge and blending. These shaders are combined to form the cartoon rendering pipeline as shown in Figure **??**. In this practical you will implement the complete cartoon rendering pipeline by writing these four shaders.

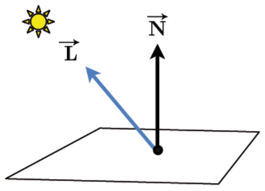
In order to setup the Visual Studio solution please follow these steps :

Go to Project Properties → Configuration Properties → Debugging
Set the Working Directory to:
```
$(SolutionDir)\..\..\src\exercises\03-CartoonViewer
```
Set Environment to:
```
PATH=%PATH%;$(SolutionDir)\..\libs
```
Set the Arguments to:
```
..\..\..\data\bunny.obj
```

Note: Download the framework onto the desktop then move it to your personal folder when you are done.

Visual Studio Solution:
```
.\visualstudio\03-CartoonViewer \03-CartoonViewer.sln
```
Shaders Directory:
```
.\src\exercises\03-CartoonViewer
```

4.2 Toon-Shading (10 points)



The main idea of the toon-shading is to replace the smooth and continuous diffuse lightning by a discrete one. To do this effect, the diffuse coefficient $0 \leq D = \max(\overrightarrow{N}.\overrightarrow{L},0) \leq 1$ computed thanks to the normal vector $\overrightarrow{N}$ and the normalized light vector $\overrightarrow{L}$ is no more used to shade the surface but is used to do a lookup in a 1D texture (similar to the Figure **??**) that store the discrete coefficients for the surface shading.

Note: D is a scalar value between 0 and 1 and hence it can be use directly as a u coordinate for the 1D texture lookup.
Note: Don't forget to multiply the shading coefficient with the color.

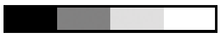


Figure 3: An example of 1D texture used for the toon-shading effect.

- **Open the file "cartoon.fs" and implement the toon-shading effect explained above. Use the "texture" sampler to get the 1D texture.**

4.3 Contour

In order to draw the contours around the different objects in black, we will first render the scene depth into a texture and then post process this texture with a Sobel operator (edge detection). The idea is to draw black pixels at depth discontinuities.

- **In order to have a visual feedback open the file "blending.fs" comment "gl_FragColor = cartoonColor;" and uncomment "gl_FragColor = edgeColor;".**

4.3.1 Linear Depth Map (2 points)

In OpenGL the depth buffer does not store the depth linearly. In our case directly using the non linear depth buffer can cause artifacts in the final rendering. To solve this issue we will implement a shader in order to output the linear depth values in a texture.

- **Open the file "depth.vs" and set the varying parameter "depth" with the depth value of the vertex re-scaled between 0 and 1. Be careful in OpenGL the camera is looking in the direction of the negative Z-axis.**

4.3.2 Edge Detection (16 points)

The edge detection will be done with a Sobel operator. The Sobel operator uses two 3x3 kernels which are convolved with the image in order to approximate the gradient of the image in X direction and the gradient of the image in Y direction.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

At each pixel (i,j) the gradient vector $G(i,j) = \begin{bmatrix} G_x(i,j) & G_y(i,j) \end{bmatrix}^T$ gives the direction of the strongest variation and its magnitude $M(i,j) = \sqrt{G_x(i,j)^2 + G_y(i,j)^2}$ indicates how strong is this variation.

- **Open the file "edge.fs", implement the Sobel operator and set the R, G and B output channels of the (i,j) fragment to $1 - 10 \times M(i,j)$. Use the uniforms dx and dy in order to move respectively from 1 pixel in x and from 1 pixel in y in the texture. Look at the values of dx and dy in the `CartoonViewer` class code and explain in the "readme.pdf" why dx and dy are set to these values.**

Note: To compute the resulting image C of the convolution of a 3x3 kernel K with an image I the 2-D discrete convolution equation is

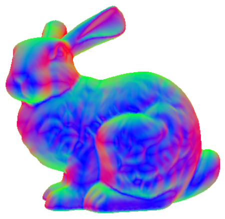
$$C(i,j) = \sum_{m=0}^{2} \sum_{n=0}^{2} K(m,n) \times I(i+m-1, j+n-1)$$

where (i,j) is the pixel position in the convolved image.

4.4 Blending (2 points)

The last operation in our cartoon rendering pipeline is blending the toon-shading image and the edge image. A simple multiplication is sufficient to blend them.

- **Open the file "blending.fs" and implement the multiplication between the toon-shading image and the edge image.**



4.5 Contours Improvement (for the passionate)*

The depth discontinuity gives already good results for a simple cartoon rendering pipeline. Nevertheless, the result can be greatly improved by adding the normal discontinuity.

- **Render the scene normals to a texture.**
- **Do an edge detection on each of the 3 layers of the normal texture and output the result on the R, G and B components of the output texture (output the depth contour on the A component).**
- **Blend the depth contour and the three normal contours with the toon-shading texture by a simple multiplication. You can try other blending and thresholding operators in order to improve the quality of the outline.**

Note: You will need to modify the code of the `CartoonViewer` class in order to bind a new texture on the FBO (for the scene normal rendering) and to bind this new texture onto the edge shader.