Prof. G. Zachmann
P. Lange

University of Bremen
School of Computer Science
CGVR Group
June 17, 2015

Summer Semester 2014

Assignment on Advanced Computer Graphics - Sheet 6

Due Date 30. 06. 2014 11:59pm
lange@cs.uni-bremen.de

Exercise 1 (Procedural Brick Texture, *4 Credits*)

In the `Advanced Shader Techniques` lecture a procedural brick texture was introduced:

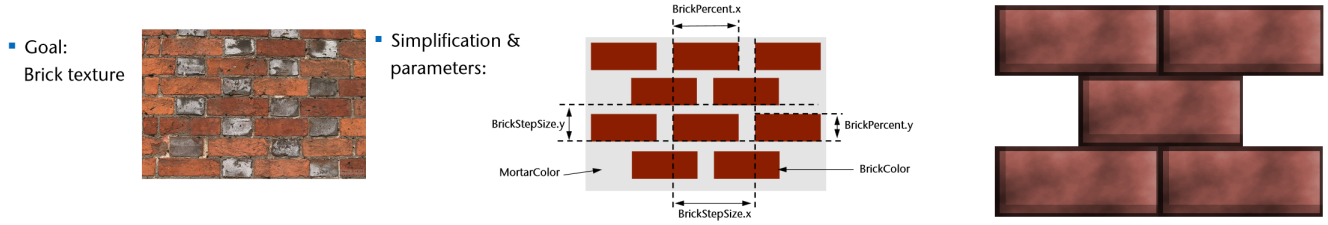

Figure 1: Procedural brick texture approach from the lecture.

Re-implement the idea in Ogre within the given framework:

- Implement the functions `BrickFramework::generateMortar` and `BrickFramework::generateBrick`, which should generate the texture for the mortar and brick respectively. For the brick and mortar texture use the `SimpleNoise` class or your favourite noise library to generate some interesting brickwalls. Also add some 'shadow' directly on the brick texture. Assume, that no lighting/shader/bump mapping/displacement/parallax mapping will be used. See the example picture for some reference of the bricks.
- Use `BrickFramework::generateBrickTexture(double BrickStepSizeX, double BrickPercentX, double BrickStepSizeY, double BrickPercentY, Ogre::TexturePtr MortarColor, Ogre::TexturePtr BrickColor)` to generate a brick texture. This function takes as input the result from your implemented `BrickFramework::generateMortar` and `BrickFramework::generateBrick` functions.
- The generated texture should be applied to the given `Ogre::Plane`, document your results with some screenshots.
- As an extension, come up with a suitable algorithm which generates circle-based bricks. Describe your algorithm (inputs, constraints, etc.) and additionally document your results with screenshots.

Exercise 2 (Culling Maths in 2D, *4 Credits*)

- Given two 2D objects O_1, O_2 each with three triangles $t_{11}((7/4),(9/2),(9/4))$ $t_{12}((9/2),(9/4),(11/4))$ $t_{13}((9/4),(9/6),(11/4))$ and $t_{21}((0/4),(1/4),(1/2))$ $t_{22}((1/2),(1/4),(3/4))$ $t_{23}((1/2),(3/4),(3/0))$. Compute the bounding spheres of O_1, O_2 and determine, whether or not O_1, O_2 are visible inside the view frustum (simplified as a box) $F : Min = (1,1), Max = (6,4)$.
- Following your bounding sphere visibility test result, determine for all triangles of a potentially visible O whether they are really inside the frustum or not. In order to do so, check for all triangles whether they are completely inside or completely outside the frustum. If a triangle is only partially visible, determine the remaining clipped triangle(s) with their vertices.
- Do a sketch of the complete scene (O_1,O_2,Frustum) and mark the rendered triangles. Write down the calculations needed to compute the culling for O_1, O_2.

Exercise 3 (Cube Maps, *2 Credits*)

Given the (s,t,r) texture coordinates of a vertex, assume that $|s|$ is the largest component by value, i.e., OpenGL will have to project (s,t,r) onto the side $x = 1$ of the unit cube (which is the parameter domain for cube maps).

Write down the GPU calculations needed to compute the (u,v) pair on that cube side.