Page Curling in WebGL

Seth Samuel*

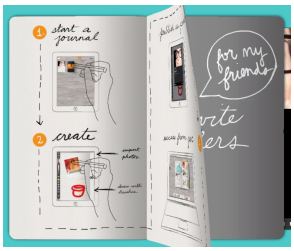

Figure 1: *WebGL page turning in Chrome 18 for OS X.*

Abstract

1 Introduction

Our application implements the page turning algorithm specified by [Hong et al. 2004] and implemented in OpenGL by [Nuon 2010]. Implementation also required the calculation of normals in the vertex shader to allow for different textures on each side of the transformed mesh.

2 Implementation

2.1 Page Curling

Hong et al. proposed a algorithm for realistic simulation of a turning page that could be calculated in realtime. Page curling is a combination of deformation process and a rotation process. The deformation takes the page as a plane anchored at the origin, and wraps it around a cone whose center lies on the y-axis. The rotation process then rotates the deformed mesh about the y-axis.

The shape and position of the cone varies in three phases to simulate the severe corner curling that occurs at the start, broader curving across the whole surface in the middle, and finally the wobble of settling onto the resting state. The attributes of the cone are calculated in Javascript then passed to WebGL shaders as uniforms. These attributes use sinusoidal easing functions to create a smoother and more realistic motion.

The vertex shader calculates its projection onto the cone as a function of its two-dimensional position on the page's plane. This projection is done in a function that takes as arguments the uniform parameters and the u-v coordinates of a point in the plane. As discussed below we calculate this three times for each vertex to determine the normal vector after the curling transformation.

The rotation is a simple 180-degree rotation about the y-axis. Without the deformation process the page would execute its turn in a completely rigid fashion. The combination of the deformation and rotation create a realistic motion that approximates the combination of flexibility and rigidity in paper.

*e-mail: seth@visere.com, web: http://myclibe.com

2.2 Two-Sided Textures

A single mesh is used for each page, which has a different texture for each side. To determine the correct texture for a given fragment, a normal vector is passed for each vertex.

Singe each page is a parameterized plane, it is simple to calculate the position of a vertex, and the position of a hypothetical vertex a small δ in either direction. The normal vector is then calculated as the cross product of these latter two vectors difference from the origin vertex. If v_0 is our projected vertex, then

$$v_s = project(v_0.s + \delta, v_0.t) \quad (1)$$

$$v_t = project(v_0.s, v_0.t + \delta) \quad (2)$$

$$v_{normal} = cross(v_s - v_0, v_t - v_0) \quad (3)$$

The normal vector is passed to the fragment shader, which determines if the normal is facing toward or away from the camera and picks the appropriate texture. This determination is made by taking the dot product with a unit vector on the z-axis. If the dot product is negative then the secondary texture is sampled (with u and v coordinates appropriately reversed).

3 Conclusion

WebGL offers a powerful environment for 3D presentation on the web. While libraries such as three.js can offer significant advantages, implementation can still be challenging.

References

HONG, L., CARD, S. K., , AND CHEN, J. J. 2004. Deforming pages of 3d electronic books. *Siggraph'04 Sketches*.

NUON, W. D., 2010. Implementing ibooks page curling using a conical deformation algorithm. `http://wdnuon.blogspot.com/2010/05/implementing-ibooks-page-curling-using.html`.