# Real-Time Bump Map Deformations

Pawel Wrotek* Brown University

Alexander Rice† Brown University

Morgan McGuire‡ Brown University

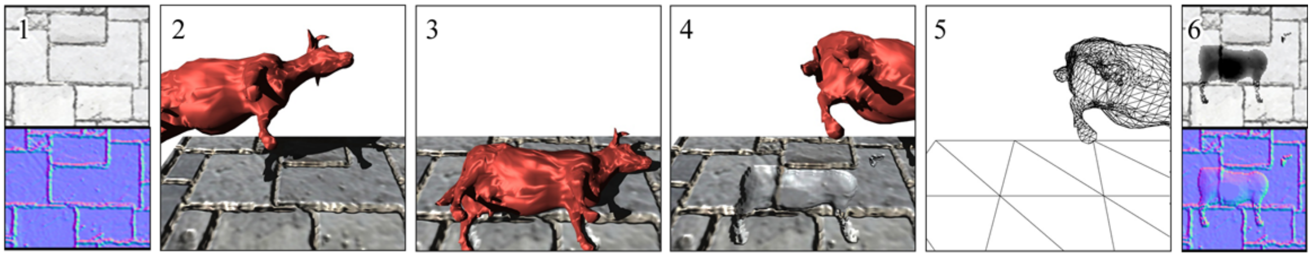

Figure 1: 1. Initial bump and normal maps for the floor. 2-4. Cow collides with floor. 5. Geometry is unchanged. 6. Floor bump and normal maps now contain a cow-shaped dent. The cow's bump and normal maps (not shown) are also flattened and deformed by the floor texture.

## Abstract

We present a method for efficiently generating plausible dents and scratches due to collisions using bump maps instead of mesh deformation. We use a rigid body simulator based on that of Guendelman et al. [2003], with collisions detected by interpenetration using the OPCODE and G3D libraries. When a collision occurs, we make multiple rendering passes to compute the bump map deformation on the GPU. Our method is limited by the dynamic range of the bump maps and will eventually saturate.

## 1 Parameterization and Rendering

As a pre-process, we create a parameterization for each object that provides a 1:1 mapping from points on the object to points on the bump map so that each point may be deformed independently during simulation. We use Sheffer and de Sturler's parameterization [2001], which minimizes shear and provides roughly uniform bump map resolution across the surface. The gradient of the bump map is Sobel filtered to produce a smooth normal map.

Objects are rendered in real-time with *parallax bump mapping*, a recent hardware-rendering trick that approximates both self-occlusion and shading for a rough surface.

## 2 Deforming Bump Maps on Collision

When objects $A$ and $B$ collide, their momentum is altered as if they were rigid bodies but their bump maps are deformed as if they were malleable. The location and shape of the deformation is computed by rendering their overlap to an offscreen depth-buffer as follows.

Place an orthographic camera a small distance along $A$'s collision normal, facing towards the collision location (Figure 2.2). Execute the following steps (with no lighting or parallax bump mapping):

1. Clear the frame buffer.
2. Render the front faces of $A$ but substitute for the color texture an *address map*, which has color $(r, g, 0)$ at texel $(r, g)$.

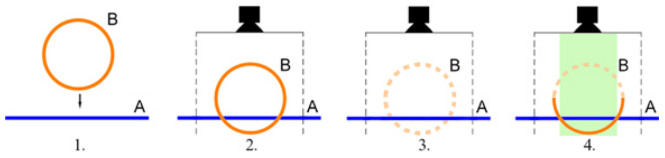

Figure 2: (1) $A$ and $B$ prior to collision. (2) Orthographic camera setup used to compute deformations on the GPU. (3) Front faces of $A$. (4) Back faces of $B$. Green box represents pixels where stencil buffer is set to 1, i.e. where $B$ penetrates $A$.

3. Read back the depth buffer $D_0$ (which now holds the "highest" points on $A$) and the color buffer $C_A$.
4. Set the depth test to pass when the new pixel is farther from the camera than the old one (GL_GREATER).
5. Render the *back faces* of $B$ using the address map, and set the stencil buffer to 1 wherever the depth test passes (i.e. wherever $B$ penetrates $A$.)
6. Read back the depth buffer $D_1$ (containing the "lowest" points on $B$), the color buffer $C_B$, and the stencil buffer $S$.

The color buffers tell us which bump map locations correspond to penetration locations, e.g., where $C_A[i, j] = (r, g, b)$, pixel $(i, j)$ was rendered by texel $(r, g)$. The difference $\Delta D(i, j) = D_1(i, j) - D_0(i, j)$ measures $A$'s geometry penetration into $B$ at this location[1].

To actually modify the bump map, we iterate over each pixel $C_A[i, j] = (r, g, b)$ on the CPU, and, if $S[i, j] = 1$ and texel $(r, g)$ in the bump map has not yet been modified for this collision, we indent it by $\Delta D[i, j]$. This process is repeated for $C_B$ and $B$'s bump map. Finally, we recompute the normal map. Modifying the bump maps on the GPU is an open problem; the challenge is ensuring that the net change to a bump texel is independent of the area it affects during rendering.

## References

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Nonconvex rigid bodies with stacking. *ACM Trans. Graph. 22*, 3, 871–878.

SHEFFER, A., AND DE STURLER, E. 2001. Parameterization of faceted surfaces for meshing using angle based flattening. *Engineering with Computers 17*, 3, 326–337.

*e-mail: pwrotek@cs.brown.edu
†e-mail: acrice@cs.brown.edu
‡e-mail: morgan@cs.brown.edu

[1] We also add the difference in bump maps so that a bump map feature can create a dent. As future work, we will write to the *gl_FragDepth* register to modify the depth maps as they are rendered in steps 2 and 5.