Real-time Minor Deformations that Result from Collisions Using Bump and Normal Mapping

Hassán Lombera Rodríguez.
University of Computer Science, Cuba.
✉ hlombera@uci.cu

Andy Trujillo Rivero
University of Computer Science, Cuba.
✉ arivero@uci.cu

ABSTRACT

This paper presents a method for simulating minor deformations that result from collisions on objects' surfaces. The method alters only bump maps and leaves mesh geometry unchanged; it is suited to real-time applications where the primary concern is computational efficiency. The paper provides a representative model for deformable objects. Texture mapping and computer graphics techniques based on lighting are referenced as well. Finally, results are provided, along with the most noteworthy findings obtained with the use of this method.

KEYWORDS: bump mapping, normal mapping, minor deformations, real-time.

Deformable object simulations can considerably increase the acceptance of many applications in computer graphics, but they have traditionally been very time consuming and difficult to execute in real-time. Therefore, programmers have placed their attention in the GPU for optimizing the scene rendering process.

On the other hand, as long as a simulation looks realistic, simplifications are deemed acceptable. Hence, this paper uses a simplified deformation model based on the stress-strain curve for simulating minor deformations that result from collisions on objects' surfaces. It is important to note that this paper only considers permanent deformations, which are represented by maintaining the geometry unchanged. To this end, techniques like bump and normal mapping are applied.

Related Work

Point-based representations for both the surface and volume of deformable objects have been researched in recent year; since they do not store the connectivity explicitly, they can handle topological changes easier (Keiser, Müller, Heidelberger, Teschner, & Gross, 2004).

For deformable objects Kaufmann proposed a simulation technique for elastically deforming objects based on the discontinuous Galerkin finite element method (Kaufmann, Martin, Botsch, & Gross, 2008). Teschner discussed various collision detection approaches that addressed several challenging problems that complicated the simulation of dynamically deformable objects (Teschner, Kimmerle, & Heidelberger, 2005). In both cases, simulations were performed off-line.

Concerning deformations in applications with real-time interaction, predefined mesh-based representations and particles systems have been merged. The latter has been even proposed for addressing structural engineering problems (Martini, 2002). Many methods have been proposed in literature, and arranged according to the level of details of deformations, in an effort to improve efficiency. Model shape representations for real-time 3D rendering commonly have three levels of detail: polygons for macro-structure, bump and normal maps for meso-structure, and BRDF approximations for micro-structure. Minor deformations affect meso-structure; here it is very important to mention a work of Morgan McGuire, who was the first to achieve minor deformations while leaving geometry unchanged (McGuire, Wrotek, & Rice, 2005). In addition, Mosegaard proposed a way of achieving real-time deformations of detailed geometry based on mapping and GPU (Mosegaard & Sørensen, 2005).

Deformation Model

In computer graphics, the more accurate a simulation of a deformation is the more time consuming it will be. But sometimes it might be convenient to sacrifice accuracy in favor of plausible results achieved in a reasonable amount of time. In this paper we will use a simplified deformation model of solid objects based on the stress-strain curve taken from Crandall's book, *An Introduction to the Mechanics of Solids*, and widely used for this purpose, see figure 1 (Fig. 1) (Crandall & Lardner, 1999).

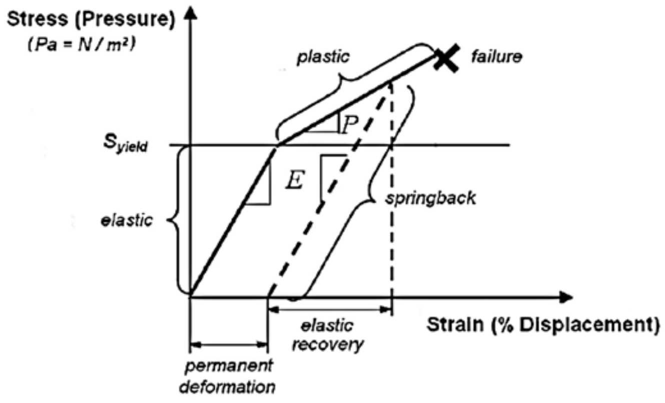

Figure 1. The simplified stress-strain curve used in this paper

As shown in figure 1 the slope of the curve during the elastic deformation phase is Young's elastic modulus *E* (Fig. 1). The plastic deformation curve is modeled as a line with a shallower slope, P < E, although materials do exist that have nonlinear responses (Gere & Goodno, 2008). So, let PD_{Total} be the total percent of deformation in response to a stress value, after some algebra operations figure 1 shows that:

$$PD_{Total} = \begin{cases} (S-S_{yield}) * \left(\frac{1}{P} - \frac{1}{E}\right), & S > S_{yield} \\ 0, & S \leq S_{yield} \end{cases} \qquad (1)$$

The parameters *E*, S_{yield} and *P* are material dependent and are available from online resources. To compute deformations, we must reconcile the mechanical engineer's stress (N/m²) model with the instantaneous impulse (N*s) collision model used for real-time simulation:

$$J = -N_C \frac{[1+\min(\varepsilon_A, \varepsilon_B)]\, N_C (\nu_B - \nu_A)}{m_A^{-1} + m_B^{-1}} \qquad (2)$$

Where NC is the collision normal, ε is the coefficient of restitution, v is velocity, and m is mass (Guendelman, Bridson, & Fedkiw, 2003). To work with these two models, we use the trends of the former but not its actual units and constants. The elastic deformation and the yield stress are represented as an impulse threshold *Y* below which no deformation occurs, and a scale factor *k* that abstracts the term $\frac{1}{P} - \frac{1}{E}$. So,

$$PD_{Total} = (j - Y_i) * \left(\frac{1}{P} - \frac{1}{E}\right) \qquad (3)$$

Putting the *k* factor in equation (3) we get the final governing equation.

$$PD_{Total} = k_A \max(0, \| j_{\varepsilon=1} \| - Y_A) \qquad (4)$$

Where $J_{\varepsilon=1}$ is the impulse that would be experienced by object *A* due to object *B* if the collision was perfectly elastic. Y_A is defined as follows:

$$Y_A = \frac{1}{1 - \varepsilon_A} - 1 \qquad (5)$$

Physical Simulation

For the physical simulation and collision detection we used Open Dynamics Engine (ODE) which was really powerful in determining the parameters needed to compute a deformation: the world space collision location P_C, the collision normal N_C, and penetration depth d_C. During the collision detection phase of the simulation we applied the technique proposed to deform colliding objects.

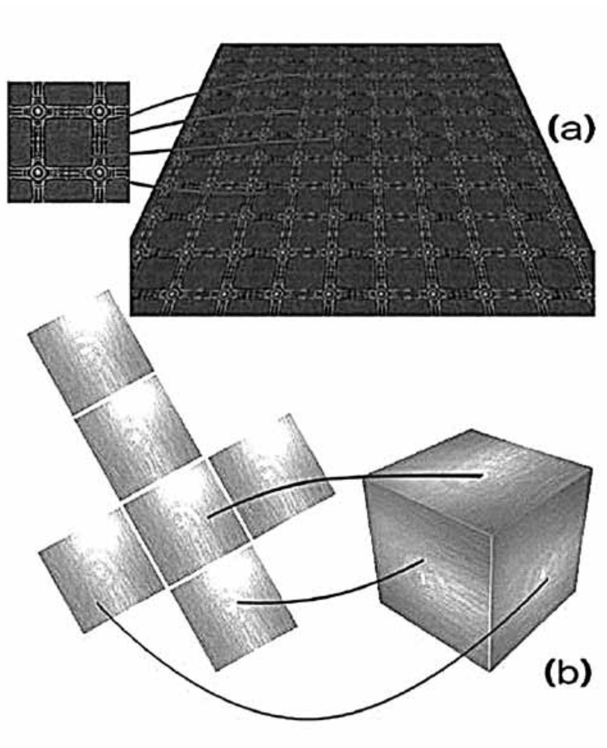

Figure 2. a) Inadequate texture parameterization b) recommended texture parameterization

Texture Parameterization

Artists usually use texture memory efficiently by tiling maps or re-using patches, see figure 2 (Fig. 2a). If such maps are applied along with the same bump map to the entire surface, we will end up having several points of the geometry connected with a unique texel, and during a collision, all of them will be affected. To ensure good performance of this method, we recommend a bump map and a texture for each object, see figure 2b (Fig. 2b).

Deforming an Object on Collision

Each object is deformed when the impulse of a collision is beyond the threshold defined in equation 5, which assures that the impact is strong enough to leave a mark. For a deformation to be plausible, it must reflect the size, shape, and elasticity of the object that caused it. To approximate this, this method causes the objects to interpenetrate during collision, and then measures the shape of the resulting overlap by using the GPU. The method also scales the depth of the deformation by the result of the stress-strain model, taking into account the coefficient of restitution of the object being deformed. This paper follows McGuire's method, which extended the one proposed by Akeley with an address map (Akeley & Jermoluk, 1988).

Computing Deformations

The computation of the deformation is executed by using shaders while rendering the back buffer objects on collision in serial form. See the steps in figure 3 for the case of a sphere colliding with a plane (Fig. 3).

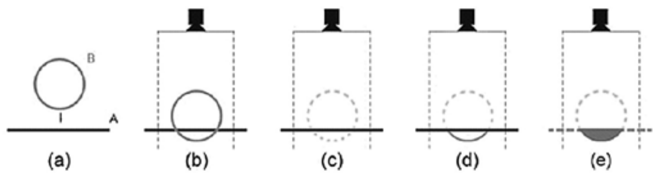

Figure 3. Steps for computing a deformation

Steps for computing the deformation for object A:

- (a) A plane *A* and a ball *B* prior to collision.
- (b) Orthographic camera setup used to compute deformations on the GPU.
- (c) Front faces of plane *A* rendered.
- (d) Back faces of ball *B* rendered where they are deeper than the previously rendered faces of plane *A*.
- (e) The resulting deformation on the surface of the plane after subtracting the calculated depths.

Then, the method retracts the object *A* by d_C along the N_C to the initial point of contact, and repeats the process for object *B*.

Updating the Bump Map

This updating process is accomplished using an address map, which is built when rendering the two objects to their color buffers. The color of a pixel is thus the 2D address of the bump map texel that corresponds to it. The blue channel is always 0. The following steps are executed with no lighting:

- Clear the frame buffer with the blue color.
- Render the front faces of *A* (Figure3c) with color = texture coordinate.
- Read back depth buffer D_A and color buffer C_A.
- Clear the color buffer and set the depth test to pass when the new pixel is farther from the camera than the old one.
- Render the back faces of B with color = texture coordinate.
- Read back the depth buffer DB.

Each object has one buffer as an address map, and the other one for storing the depth of the deformation. The difference between depth buffers, in this example, stands for the distance object *B* penetrates into object *A*. Since the objects are bump mapped, the deformation should reflect not only the objects' geometries, but also the information from those bump maps. Therefore, a fragment shader is used to for alter the values that are put into the depth buffers. For each pixel, this shader uses the corresponding value from the address map to index the object's bump map, adjusts the pixel's depth by the resulting height value, and sets the final depth accordingly. To alter the depth buffers based on the objects' bump maps, the values of the latter must be converted into depth values. This method only sets the bump map updated version of an object once the whole process has been repeated for the other one.

Using Normal Mapping

When the bump map has been updated for each object, in order to add the correct details to shading without using more polygons, the renderer would need to compute, frame by frame, appropriate normals by comparing each pixel with the ones in its vicinity. So, in favor of speed it would be much better if the normal mapping technique was used, because it stores the direction of a surface normal by using the three color channels. Therefore, the calculations of the slopes for each pixel just need to be performed one time or at least when a deformation occurs. That is why this paper includes this technique in the approach proposed by McGuire mentioned earlier. As the computation of the normal map can be very time consuming depending on the image resolution, we use the same address map technique for its updating process.

Results and Discussion

The method this paper proposes, although it is physically motivated, provides different results from physically correct ones. Despite this, results are perfectly suitable for real-time applications where efficiency and robustness are desired. In addition, deformations keep the geometry unchanged. The use of the simplified deformation model allowed plausible representations of minor deformations on rigid bodies' surfaces. The use of the GPU througout the method also contributed to increased speed of the process. On the other hand, large-scale deformations that change an object's overall structure cannot be performed by this method, since it does not alter an object's mesh. In that case, would be convenient to use a hybrid method which combines the one proposed here for minor deformations with another method; research of that subject is left for future work. Moreover, the method avoids unnecessary calculations if collisions do not produce permanent deformations on object's surface. The main contribution of this method is the incorporation of the normal mapping technique in the one proposed by McGuire. This inclusion reduces the continuous normals calculations after a bump map is updated. The updating process occurs fully in the CPU, because current GPU technology allows for texture lookups of up to four textures within the vertex shader, which is not enough for the purpose of this method. This method needs five texture lookups for each rendering pass. For the implementation of the demo, Graphics Three-Dimensional engine (G3D) was used. See figure 4 for a snapshot of the created demo (Fig. 4).

The method could also be integrated with the physics engine ODE, since it has only been used with the simulator proposed by Guendelman (Guendelman, Bridson, & Fedkiw, 2003).

Regarding the design process one may note that graphic artists dealing with Computer Aided Design software are commonly concerned with the way virtual models being developed react to external physical stimulus, so that they can improve its design and reduce costs.

These kinds of results are typically obtained by applying complex numerical techniques like the finite element method (FEM), which is very accurate but also time consuming. However, for some applications like videogames, precision is not that important; what is sought is a little bit of realism. Due to the interactive and real-time nature of these applications, complex methods like FEM are hard to apply. Here is where our proposal could be used.

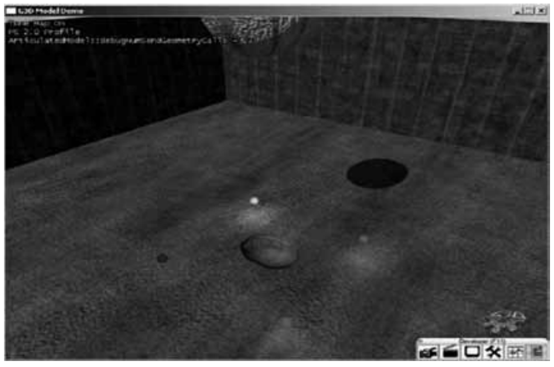

Figure 4. A snapshot of the created demo

This method could also be included in CAD modeling tools as an add-in which would allow artists to check how their models will behave under different stimuli. By adjusting the parameters previously discussed, they could gain an idea of how those changes could affect the solution. Virtual laboratories can also use this technique to allow students to experiment real-time interactions when learning physical sciences.

Conclusions

After the completion of this research we concluded that: the use of this method guarantees the computation of plausible deformations and keeps the objects' meshes unchanged; the use of the GPU for the deformation computation guarantees to speed up the calculation process; the addition of the normal mapping technique to the method proposed by McGuire avoids the continuous normals calculations while rendering the scene; the results are widely applicable to real-time applications where the most important aspect is the generation of plausible behaviors, rather than the accurate predictions of exact results; the method can be re-created regardless of the graphics or physics engines chosen for this purpose.

References

Akeley, K. & Jermoluk, T. (1988). *High-performance polygon rendering*. Proceedings of the 15th annual conference on Computer graphics and interactive techniques, ACM Press, pp. 239-246.

Crandall, S. H. & Lardner, T. J. (1999). *An Introduction to the Mechanics of Solids* (2nd ed.). New York: McGraw-Hill.

Gere, J. M. & Goodno, B. J. (2008). *Mechanics of Materials*. Thomson Engineering.

Guendelman, E., Bridson, R., & Fedkiw, R. (2003). *Nonconvex Rigid Bodies with Stacking*. Los Angeles: ACM Trans.

Kaufmann, P., Martin, S., Botsch, M., & Gross, M. (2008). *Flexible Simulation of Deformable Models Using Discontinuous Galerkin FEM*. Zurich: Eurographics.

Keiser, R., Müller, M., Heidelberger, B., Teschner, M., & Gross, M. (2004). *Contact Handling for Deformable Point-Based Objects*. Stanford.

Martini, K. (2002). A Particle-System Approach to Real-Time Non-Linear Analysis. Virginia: Proceedings of the 7th National Conference on Earthquake Engineering [CD-ROM], Earthquake Engineering Research Institute.

McGuire, M., Wrotek, P., & Rice, A. (2005). *Real-Time Collision Deformations using Graphics*. Boston: A K Peters.

Mosegaard, J. & Sørensen, T. S. (2005). *Real-time Deformation of Detailed Geometry Based on Mappings to a Less Detailed Physical Simulation on the GPU*. Proceedings of Eurographics Workshop on Virtual Environments, vol. 11, pp. 105-111.

Teschner, M., Kimmerle, S., & Heidelberger, B. (2005). Collision Detection for Deformable Objects. Computer. *Graphics Forum*, 24, 61-81.