Example-based Plastic Deformation of Rigid Bodies

Seminar: Current Topics in Physically-Based Animation

Younes Müller

January 15, 2017

Contents

1 Introduction

In this seminar work I will describe and explain the method described by Ben Jones, Niles Thuerey, Tamar Shinar and Adam W. Brgteil in their article "Example-based Plastic Deformation of Rigid Bodies."[BJB16] The goal is to develop a fast, example based method to simulate deformation of rigid bodies. This has the advantage that the user of the method can control, how the meshes look after deformation. This is achieved through combining linear blend skinning with example based deformation.

2 Method

An artist provides a set of example poses to show the simulator, how the body can transform on impact. It is not necessary to provide a lot of poses, but the method is designed for few examples. If the object does not behave as desired more poses can be added. Example based deformation opens the opportunity to simulate the models with a rigid body simulator. This is suitable for near rigid bodies, as the internal movement and vibration is too fast to be noticed. Furthermore stiff materials have very high forces inside, which gets unstable very fast, if simulated through mass spring systems.

We are using a mesh with N vertices.

2.1 Rigid bodies

To simulate the translational motion of a rigid body it is sufficient to simulate the motion of the center of gravity. As the vertices stay motionless relative to the center the position of every vertex can be calculated easily from the position of the center. If the motion is arbitrary it can still be represented by just a translational and a rotational movement. The simulation is done by a rigid body simulator. To simulate the body it stores the following parameters:

- densitiy ρ and mass m
- the moment of inertia, which is constant to each body and behaves to rotational movement, as mass behaves to translational movement.
- linear position $\boldsymbol{x}$ and momentum $\boldsymbol{p}$
- orientation $\boldsymbol{\Omega}$ and angular momentum $\boldsymbol{L}$
- coefficient of restitution C_r, which stores the ratio between elastic and non elastic collision

2.2 Skinning

It is far too complex to provide new positions for every single vertex. So a method named skinning is used. The mesh is rigged with bones. Every vertex is assigned to

a bone. And similarly to real bones the vertices follow the movement of the bone. To prevent artifacts a vertex can not only be assinged to one bone, but to multiple bones with weighting.
The bones behave a bit like real bones. For example: In a model of an arm, the bones would be placed similarly to the bones in a real arm and if the bones would be moved, the vertices would follow. A vertex at the elbow would be associated with the bone of the upper and of the lower arm, so that it follows both movements.
If we have the bones B have to store some more values:

- undeformed mesh vertex positions: $\boldsymbol{u} \in \mathbb{R}^{N\times 3}$
- skinning weights: $\boldsymbol{W} \in \mathbb{R}^{N\times B}$
- the desired transformation, which consist of:
  rotation: $\boldsymbol{R} \in \mathbb{R}^{B\times 3\times 3}$
  translation: $\boldsymbol{T} \in \mathbb{R}^{B\times 3}$

The skinning weight matrix $\boldsymbol{W}$ contains a row for each vertex and a column for each bone. The Value w_{ib} says how much percent of the movement of bone b affects vertex $\boldsymbol{x}$.

So to get the transformed position of a single vertex we apply the weighted transformation of each bone on the vertex.

$$\boldsymbol{x}_i = \sum_{b\in B} \boldsymbol{W}_{ib} \left(\boldsymbol{T}_b + \boldsymbol{R}_b \boldsymbol{u}_i\right) \tag{1}$$

As we want to use multiple example poses on every vertex we don't have one desired Transformation, but one for every pose, generated by the artist. So we extend $\boldsymbol{R}$ and $\boldsymbol{T}$ and add a matrix that contains, how much every example pose acts on every vertex:

- $\boldsymbol{R} \in \mathbb{R}^{B\times E\times 3\times 3}$
- $\boldsymbol{T} \in \mathbb{R}^{B\times E\times 3}$
- $\boldsymbol{E} \in \mathbb{R}^{N\times E}$

We have to somehow consider mutliple example poses. The native approach is to use linear interpolation and weight the individual transformations with the weighting matrix $\boldsymbol{E}$.
We now can determine the new position of each vertex with:

$$\boldsymbol{x}_i = \sum_{b\in B} \boldsymbol{W}_{ib} \left(\sum_{e=1}^{E} \boldsymbol{E}_{ie} \left(\boldsymbol{T}_b + \boldsymbol{R}_{be} \boldsymbol{u}_i\right) \right) \tag{2}$$

With the translational part this works fairly well, however linear interpolation does not work well on rotation matrices. This is due to the fact, that a 3 dimensional rotation has 3 degrees of freedom, while a rotation matrix has 9 entries. So 6 Entries are somehow

redundant and have to be bound via constraints (orhogonality and determinant is 1). So if we linearly interpolate each single entry of a roation matrix it is not guaranteed, that the resulting matrix is itself a rotation matrix. To solve this quaternion rotation is used.

2.2.1 Quaternion rotation and QLERP

Ken Shoemake found out that a 3 dimensional rotation can be represented by a quaternion. Quaternions are an extension of the complex numbers. They have 3 imaginary units and are based on the identity

$$i^2 = j^2 = k^2 = ijk = -1 \tag{3}$$

Each Quaternion has the form: $a + bi + cj + dk$.How exactly a quaternion can be used to rotate a vector can be read in [Sho85]. Quaternion rotation has a main advantages over matrix rotation. We only have one additional entry to the 3 degress of freedom in a quaternion. So only one constraint is needed to provide a valid rotation. It turns out that a rotation quaternion has to be of length one.
So to interpolate between two rotations we can interpolate the corresponding rotation quaternions and normalize them. [Kr05] The formula for interpolation of the rotation quaternions $\boldsymbol{q}$ and $\boldsymbol{p}$ with the parameter $t \in [0, 1]$ the following formula is used:

$$l(t; \boldsymbol{p}, \boldsymbol{q}) = \frac{(1-t)\boldsymbol{p} + t\boldsymbol{q}}{||(1-t)\boldsymbol{p} + t\boldsymbol{q}||} \tag{4}$$

To blend multiple rotations we now use a matrix, containing the quaternion values for each example pose $\boldsymbol{R}^{4\times E}$ and the weightings $\boldsymbol{E}^{N\times E}$. This works, because the columns of $\boldsymbol{E}$ sum up to 1.

$$\boldsymbol{R}'_i = \text{QLERP}(\boldsymbol{E}_i, \boldsymbol{R}) = \text{normalize}\left(\sum_{e\in E} \boldsymbol{E}_{ie}\boldsymbol{R}_e\right) \tag{5}$$

2.3 Recap Skinning

Now we have a method to deform meshes with the help of bones and to blend in given example poses. The input consists the constant data:

- The Vertex positions in undeformed state: $\boldsymbol{u} \in \mathbb{R}^{N\times 3}$
- The weighting of each bone for the vertcies: $\boldsymbol{W} \in \mathbb{R}^{N\times B}$
- The transformation for each example pose, consiting of

  The rotational part $\boldsymbol{R} \in \mathbb{R}^{B\times 4\times E}$

  The translational part $\boldsymbol{T} \in \mathbb{R}^{B\times 3\times E}$

and the changing data:

Ist das hier okay, dass ich die Notation der Matritzen missbrauche? Der erste Parameter der Rotationatrix ist ja eigentlich, welches Element eines Quaternions verwendet wird, aber aus der Notation wird denke ich klar, dass statdessen der einem Beispiel zugeordnete quaternion

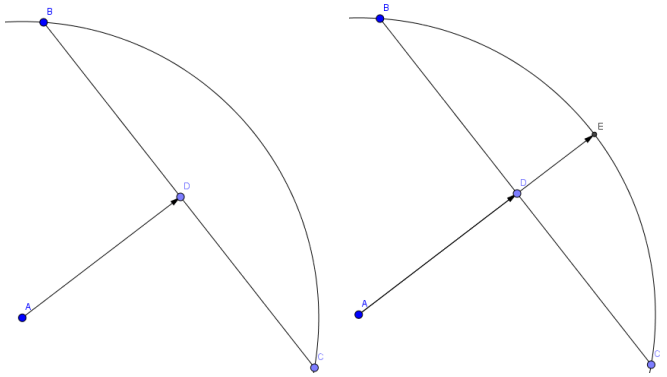

Figure 1: On the left: Idea, how linear interpolation between rotation matrices looks. If linear interpolation between two points on a circle is used, the resulting points do not lie on the circle. On the right: Interpolating with quaternions. The result can be easily mapped onto the sphere by normalizing the quaternion.

- example Weights $\boldsymbol{E} \in \mathbb{R}^{N \times E}$

Let rotate$(\boldsymbol{q}, \boldsymbol{x})$ be the function that rotates the position $\boldsymbol{x}$ by the quaternion $\boldsymbol{q}$. The skinned mesh vertex positions $\boldsymbol{x} \in \mathbb{R}^{N \times 3}$ can now be calculated by:

$$\boldsymbol{x}_i = \sum_{b \in B} \boldsymbol{W}_{ib} \left(\sum_{e=1}^{E} \boldsymbol{E}_{ie} \boldsymbol{T}_{be} + \text{rotate}(\text{QLERP}(\boldsymbol{E}_i, \boldsymbol{R}_b), \boldsymbol{u}_i) \right) \tag{6}$$

2.4 Calculate example weighting

When two objects collide, the rigid body simulator passes the vertex $\boldsymbol{x}_i$, where the collision occurred and the impulse $\boldsymbol{j}_i$. The goal is now to calculate a useful configuration of the example weights, that matches the impact. As example poses are used, not every deformation is possible. So we first calculate an ideal deformation for the vertex being hit. Then we solve a minimization problem, trying to get good example weightings, so that the distance between the ideal deformation and the skinned position of the vertex is minimal. Afterwards we propagate this to the vertices around.

2.4.1 Projection

The formula for the ideal deformation is given by:

$$\boldsymbol{\Delta x}_i = \frac{\Delta t}{m} \max(||\boldsymbol{j}_i|| - \beta, 0) \frac{\boldsymbol{j}_i}{||\boldsymbol{j}_i||} \tag{7}$$

It extends the usual formula for the deformation $\boldsymbol{\Delta x}_i = \frac{\Delta t}{m} \boldsymbol{j}_i$ with the user adjustable threshold β, on how strong the impact has to be at minimum to be registered. This

prevents deformation on resting contact. The term Δt is the size of a timestep and m is the mass of the object. We now construct the Jacobian matrix $\boldsymbol{J}$ by

$$\boldsymbol{J}_i = \frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{E}_{ie}} \in \mathbb{R}^{3\times E} \tag{8}$$

It contains the change in vertex position depending on a change in the example weighting. So given a change of position Δx, in the example space we can compute the change of example weightings by:

$$\boldsymbol{\Delta e}_i = \boldsymbol{J}_i^T \boldsymbol{\Delta x}_i \tag{9}$$

This formula however requires the change to be in the space of possible changes, that can be achieved by the examples. To get a good $\boldsymbol{\Delta e}$ we have to minimmize the following problem:

$$\boldsymbol{\Delta e} = \min_{\boldsymbol{\Delta \hat{e}}} ||(\boldsymbol{x}_i(\boldsymbol{e} + \boldsymbol{\Delta \hat{e}}) - \boldsymbol{x}_i(\boldsymbol{e})) - \boldsymbol{\Delta x}_i||^2 \tag{10}$$

A method to solve minimization problems is the method of steepest descent. The direction of the gradients, which points in the direction of steepest ascent on every point on the (hyper-)plane is used to iteratively go in the direction of the steepest descent to search a lokal minimum. As we do not opt for physical accuracy it is sufficient to go one step. As initial value $\boldsymbol{\Delta e} = 0$ is used. The gradients can be calculated by transposing the Jacobi matrix:

$$\frac{\boldsymbol{J}_i^T \boldsymbol{\Delta x}_i}{||\boldsymbol{J}_i^T \boldsymbol{\Delta x}_i||} \tag{11}$$

is the direction of the gradient on Δx

The stepwidth of the step is chosen, so that it is dependent on the impact and user controllable. The approximated change in Example weightings is calculated by:

$$\boldsymbol{\Delta e}_i = \alpha ||\boldsymbol{\Delta x}|| \frac{\boldsymbol{J}_i^T \boldsymbol{\Delta x}_i}{||\boldsymbol{J}_i^T \boldsymbol{\Delta x}_i||} \tag{12}$$

2.5 Propagation

If we would apply the change in example weights only on the vertices, that were hit, the result would look bumpy. So the change in example weights is propagated to nearby vertices. As distance measure, the approximated geodesic distance via a few Dijkstra iteration is used. The change of Example weights for every vertex is saved in the Matrix $\boldsymbol{\Delta E}$. To propagate the change we modify each column by:

$$\boldsymbol{\Delta E}_j \mathrel{+}= \phi\left(\frac{||\boldsymbol{x}_j - \boldsymbol{x}_i||}{\gamma}\right) \boldsymbol{\Delta e}_i \tag{13}$$

$$\phi(x) = \begin{cases} 2x^3 - 3x^2 + 1 & : x < 1 \\ 0 & : otherwise \end{cases} \tag{14}$$

This assures that the vertecies within a user determined distance γ of the impacted vertexare also affected.

2.6 Application

To let the impact look more natural, the deforming does not happen instantaneously, but over time. The Matrix $\boldsymbol{\Delta E}$ contains the change in example weighting that still has to be applicated. In every timestep the current example weighting $\boldsymbol{E}$ and $\boldsymbol{\Delta E}$ are updated via:

$$\boldsymbol{E} \mathrel{+}= (1-\lambda)\boldsymbol{\Delta E} \tag{15}$$

$$\boldsymbol{\Delta E} \mathrel{*}= \lambda \tag{16}$$

where λ is a user given parameter. For $t \to \infty$ the whole change is applicated.

2.7 Restitution Modifiaction

As a deformation consumes energy it is necessary to show this in the movement of the object. With the deformation the nonelastic part of the collision increases. The ration between the elastic and non-elastic part, the coefficient of restitution is modified by:

$$C_r := \min(C_r^*, C_r + \mu\Delta t, \exp(-\nu||\boldsymbol{\Delta E}||_f)C_r^*) \tag{17}$$

where C_r^* is the coefficient that is initially used by the rigid body simulator and μ and μ are user given.

References

[BJB16] Tamar Shinar Ben Jones, Nils Thuerey and Adam W. Bargteil. Example-based plastic deformation of rigid bodies. *ACM Transactions on Graphics*, 35(4), July 2016.

[Kr05] Ladislav Kavan and Jiří Žára. Spherical blend skinning: A real-time deformation of articulated methods. *I3D '05*, July 2005.

[Sho85] Ken Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH Computer Graphics*, 19, November 1985.