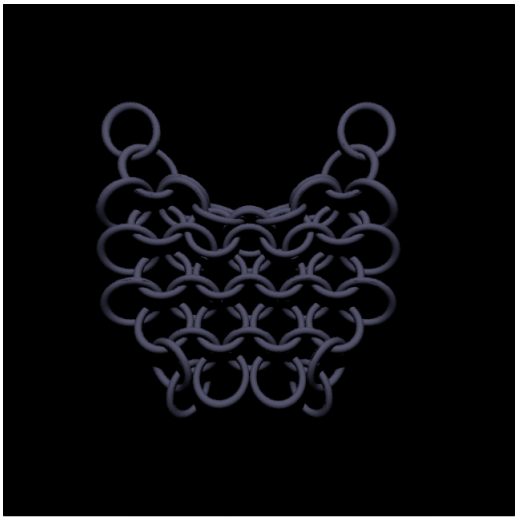
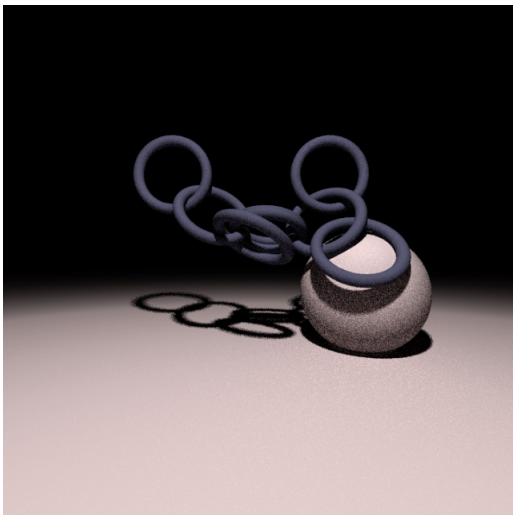

6.837 Final Project: Chainmail

Trevor Henderson

December 25, 2016

Abstract

I simulated chainmail — a type of cloth made out of linked metal rings. The links of the chainmail is stored in a layered range tree. This data structure offers $O(k + log^2 n)$ range queries, which significantly speeds up collision detection. All of the objects in my simulations are treated implicitly. My ray tracer can render arbitrary polynomial implicit surfaces. Collisions between objects are handled with penalty forces. I am using my own method to compute penalty forces between implicit surfaces using Monte Carlo integration.

Videos

5 by 10 Chainmail Drop: `https://youtu.be/8PuGiX6OHzk`
Chainmail-Sphere collision: `https://youtu.be/iL0trOxa0wY`
Bloopers: `https://youtu.be/1qtzbqadsVA`

Code

Github: `https://github.com/sportdeath/Chainmail`

1 Overview

My original intention with this project was to implement cloth with yarn based mechanics. However due to the complexity of the mechanics of yarn I realized that an implementation was outside of the scope of time I had available. So instead I decided to implement cloth by "knitting" rigid bodies as opposed to flexible ones. I chose chainmail; a type of cloth used as armor that is made out linked metal rings. In particular my simulations model European style 4 in 1 chainmail. This style was common throughout the Eurasian landmass from around the 4th century BC to the 17th century [7]. Every ring contains 4 other rings as shown in figure 1.

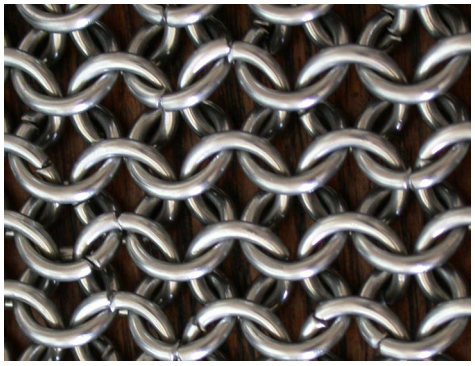

Figure 1: European style 4 in 1 chainmail

2 Layered Range Tree

A layered range tree is a data structure that can store d-dimensional data in $O(n \log^{d-1} n)$ space and perform range queries in time $O(k + \log^{d-1} n)$ time, where k is the number of elements produced by the query. For 3 dimensional data, this is an $O(k + \log^2 n)$ time query, which is faster than performing a linear search, or using kd-trees which can produce range searches in $O(n^{2/3})$ time. Additional augmentations to layered range trees can produce query times as low as $O(k + \log n)$, however this blows up the space and therefore construction time by several log factors. In collision detection you perform a linear number of range searches per time step, and at each time step the points are updated and the data structure needs to be reconstructed, so the speedup would not be worth it.

A single dimensional range tree is a binary tree where the elements are stored at leaf nodes. Range trees take up $O(n \log n)$ space and can perform queries in $O(\log n)$. A range query between a and b traverses the tree to find the predecessor leaf node of a and the successor leaf node of b and then returns the elements of all subtrees contained in between those two nodes. A layered range tree is a nesting of range trees where each node on one layer of trees points to the tree in a layer below. Each layer is sorted in a different dimension. The nodes stored in each tree correspond to the nodes stored in the subtree of the parent tree node that points to it. So to perform a search, a range query is done in the first dimension, then for all contained subtrees, range queries are done in orthogonal dimensions until at last the last layer is reached and the results are returned.

However an additional trick is applied to the last layer of the range tree. The last layer of range trees is replaced with arrays. Using a technique called fractional cascading, the last and second to last layer are traversed at the same time by following precomputed predecessor and successor pointers between the arrays. This reduces the log factor in space and range querying from $O(\log^d n)$ to $O(\log^{d-1} n)$ [3].

3 Raytracing

My implementation of ray tracing began with the starter code in pset 4 with Lambertian reflectance, shadows, and basic model rendering. In addition to techniques I added path tracing, anti-aliasing with jitter, the ability to render arbitrary polynomial implicit surfaces defined by polynomial F:

$$F(\mathbf{x}) = 0$$

For a z-axis aligned torus with major radius R and minor radius r I used the following equation [9]:

$$F(\mathbf{x}) = \left(\mathbf{x} \cdot \mathbf{x} + R^2 - r^2\right)^2 - 4R^2 \left(\mathbf{x} \cdot \mathbf{x} - (\mathbf{x} \cdot \hat{\mathbf{z}})^2\right) = 0$$

In order to compute the intersections of rays with implicit surfaces I implemented a polynomial class and a polynomial vector class. A ray for example with origin $\mathbf{o}$ and direction $\mathbf{d}$ is a vector of polynomials

$$\mathbf{r}(t) = \begin{bmatrix} d_x t + o_x \\ d_y t + o_y \\ d_z t + o_z \end{bmatrix}$$

These polynomial vector can be added, crossed and dotted like normal vectors to produce other polynomial vectors and polynomials rather than vectors and constants. I find the roots of polynomials by computing the eigenvalues of their companion matrices. For monomial $p(t) = c_0 + c_1 t + \cdots + c_{n-1}t^{n-1} + t^n$, the companion matrix is

$$\begin{bmatrix} 0 & 0 & \dots & 0 & -c_0 \\ 1 & 0 & \dots & 0 & -c_1 \\ 0 & 1 & \dots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -c_{n-1} \end{bmatrix}$$

Since the characteristic polynomial of this matrix is equal to $p(t)$ its eigenvalues are the roots of $p(t)$ [8]. Roots with imaginary coefficients greater than a certain value are filtered out.

4 Physics

4.1 Basic forces

The basic physical model I am using is based off of the Siggraph '97 course notes on rigid body dynamics [1]. In the described method every rigid body is represented by a state vector that encodes the center of mass, rotation matrix, momentum and angular momentum.

$$\mathbf{Y} = \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{R}(t) \\ \mathbf{P}(t) \\ \mathbf{L}(t) \end{bmatrix}$$

The derivative of this state encodes velocity, angular velocity, force and torque.

$$\frac{d}{dt}\mathbf{Y} = \begin{bmatrix} \mathbf{v}(t) \\ \omega(t) * \mathbf{R}(t) \\ \mathbf{F}(t) \\ \tau(t) \end{bmatrix}$$

Here the $*$ operator represents the multiplication between $\mathbf{R}(t)$ and the skew-symmetric matrix generated by $\omega(t)$.

In order to evaluate the differential equation, I am using the Runge-Kutta 4 integrator from pset 3. The basic forces I have implemented are gravity, and drag. Moment of inertia from torii are computed physically using the following equations [9]:

$$\mathbf{I} = \begin{bmatrix} \left(\frac{5}{8}r^2 + \frac{1}{2}R^2\right)m & 0 & 0 \\ 0 & \left(\frac{5}{8}r^2 + \frac{1}{2}R^2\right)m & 0 \\ 0 & 0 & \left(\frac{3}{4}r^2 + R^2\right)m \end{bmatrix}$$

4.2 Collision Handling

In order to handle collisions I am using penalty forces. This method treats the surfaces of objects as springs which apply repelling forces based on the penetration depth of one object into another [4]. Rather than impulse response methods, penalty methods work more naturally with a numerical integrator and can cost less computationally. Additionally, impulse methods require determining the time at which two objects collide, which would be particularly difficult for implicit surfaces [6]. The penalty force method does have the drawback that objects do intersect slightly, however this effect is negligible with stiff enough springs.

My method is based off of Evan Drumwright's penalty force method, which uses multiple penetration points on an object to determine the penalty force [4]. The resultant forces and torques are averaged together. For object A being acted on by object B, the force on point $\mathbf{p}_A$ by closest point $\mathbf{p}_B$ is computed by

$$\mathbf{F} = \hat{\mathbf{n}}_B(k_p||\mathbf{p}_A - \mathbf{p}_B|| - k_v||\mathbf{v}_A - \mathbf{v}_B||)$$

Where the $\hat{\mathbf{n}}_B$ is the normal at $\mathbf{p}_B$, and $\mathbf{v}_{\mathbf{p}_A} - \mathbf{v}_{\mathbf{B}_A}$ is the relative velocity of the points. The velocity of a point $\mathbf{p}_A$ is

$$\mathbf{v}_{\mathbf{p}_A} = \mathbf{v}_A + \omega_A \times (\mathbf{p}_A - \mathbf{x}_A)$$

So far as I could tell, no penalty force method handles the case of implicit surfaces. My method does the following:

Generate n random points $\mathbf{p}_A$ on the surface of object A from parametric equations.
for each point $\mathbf{p}_A$ **do**
  if $\mathbf{p}_A$ is contained by object B i.e. $F_B(\mathbf{p}_A) < 0$ **then**
    Calculate the closest point $\mathbf{p}_A$ and its normal to $\mathbf{p}_A$ on object B
    Compute the spring force using $\mathbf{p}_A$, $\mathbf{p}_B$, and $\hat{\mathbf{n}}_B$
  end if
end for

For a torus the parametric equations are [9]:

$$x(\theta, \phi) = (R + r\cos\theta)\cos\phi$$
$$y(\theta, \phi) = (R + r\cos\theta)\sin\phi$$
$$z(\theta, \phi) = r\cos\theta$$

The closest point to $\mathbf{p}$ on a torus can be computed as follows:

Project $\mathbf{p}$ onto the plane of the R-radius circle: $\mathbf{p}'$
$\mathbf{q} = R||\mathbf{p}'||$ ▷ The closest point on the R-radius circle
return $r||\mathbf{p} - \mathbf{q}||$ ▷ The closest point on the r-radius circle

Although I did not implement this, for arbitrary surfaces, I suspect the closest point on the surface could be estimated with gradient descent.

I implemented frictional contact by adding virtual springs in the plane tangent to the relative velocity and surface normal.

5 Implementation Results

I implimented this project in C++ using starter code from psets 3 and 4 in addition to the libraries Eigen [5] and OpenCV [2]. Eigen was used for matrix and vector operations, and OpenCV was used for adding matrices of color values to a movie file.

5.1 Range Tree

The asymmetry of the layered range tree made it particularly difficult to implement while also staying as abstract as possible However, after many

segfaults it finally passes a comprehensive test suite and works well in the simulations. With 4 in 1 chainmail the range tree checked an average of 10 neighboring torii per torus and did not grow with the dimensions of the chainmail.

The range tree query time was practically negligible to computation. I did not realize until the final stages of my project how little I would be able to render, otherwise I would have come up with some other way to show off the tree. If however I made rendering faster, the tree would make scaling up practically linear.

5.2 Ray Tracing

The results of path tracing are shown in figure 2. Ray tracing was by far the slowest part of computation. From a profiler I measured that almost all of the ray tracing computation time the root finding process which is bounded by how fast the Eigen library can solve for the eigenvalues of a matrix. I bounded all of my torii in spheres of radius $r + R$, which reduced ray tracing time greatly but it still lacks in efficiency. Were I too continue work on this I would first approximate the implicit surfaces with polygons before rendering them.

As a result of this slowdown, I could not run large simulations efficiently and hence the true power of the layered range tree was never utilized.

5.3 Physics

I made a chainmail class which generates a sheet of 4 in 1 chainmail with an arbitrary number of rows and columns. With this, I made several simulations. I believe the physical parameters still need to be tuned for physical accuracy; there is generally too much drag and a slight amount of bouncing from the stiff springs. But overall I believe it looks and hangs like chainmail should!

Results are shown in figures 3 and 4, and video links can be found here:

5 by 10 Chainmail Drop: `https://youtu.be/8PuGiX6OHzk`
Chainmail-Sphere collision: `https://youtu.be/iL0trOxa0wY`
Bloopers: `https://youtu.be/1qtzbqadsVA`

References

[1] David Baraffi. An introduction to physically based modeling: Rigid body simulation iunconstrained rigid body dynamics. Siggraph'97 Course Notes, 1997. https://www.cs.cmu.edu/~baraff/sigcourse/notesd1.pdf.

[2] G. Bradski. Opencv. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Erik Demain. 6.851: Advanced data structures, lecture 3, Spring 2012. https://courses.csail.mit.edu/6.851/spring12/lectures/L03.html.

[4] Evan Drumwright. A fast and stable penalty method for rigid body simulation, 2008. http://www-robotics.usc.edu/~drumwrig/pubs/tvcg.pdf.

[5] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[6] James K. Hahn. Realistic animation of rigid bodies. ACM, 1988. http://dl.acm.org/citation.cfm?id=378530.

[7] David Henderson. Costume history discussion, 2016.

[8] Charles R. Johnson Roger A. Horn. Matrix analysis. Cambridge University Press New York, NY, USA, 1986.

[9] Eric W. Weisstein. Torus. Mathworld. http://mathworld.wolfram.com/Torus.html.

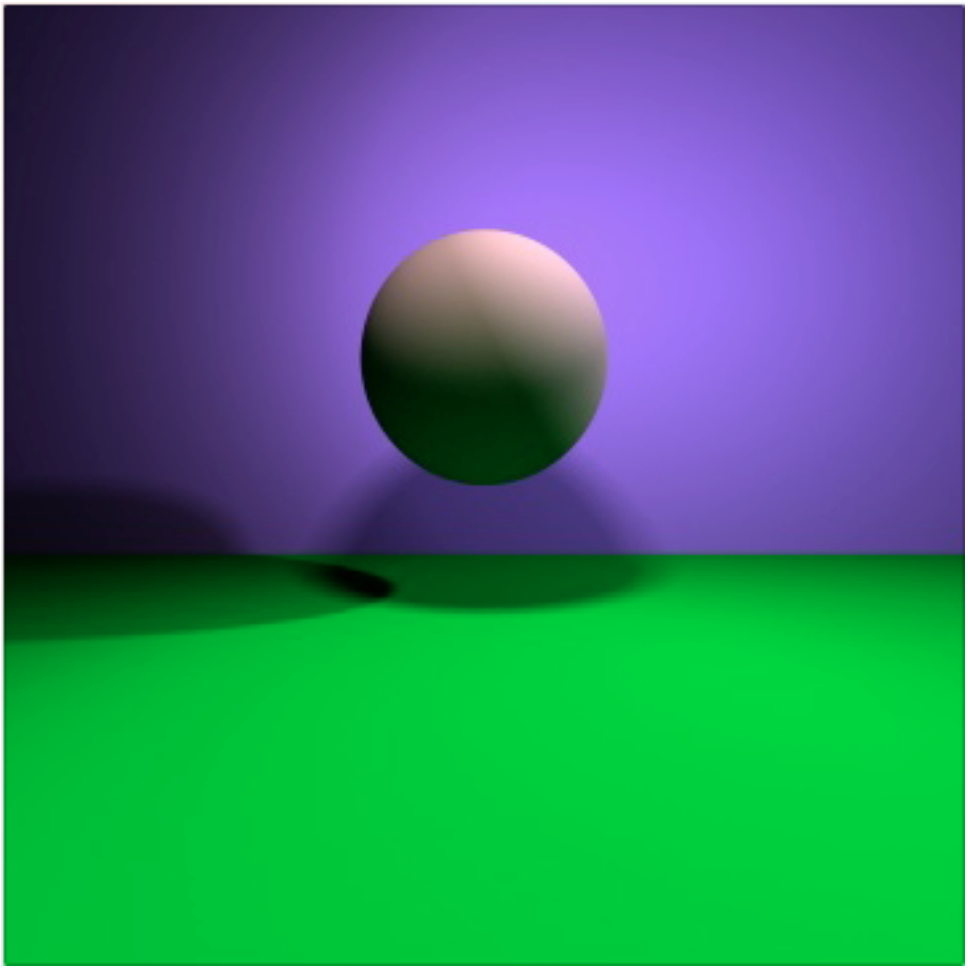

Figure 2: A ray traced sphere with 2 bounced and an anti-aliasing factor of 1024

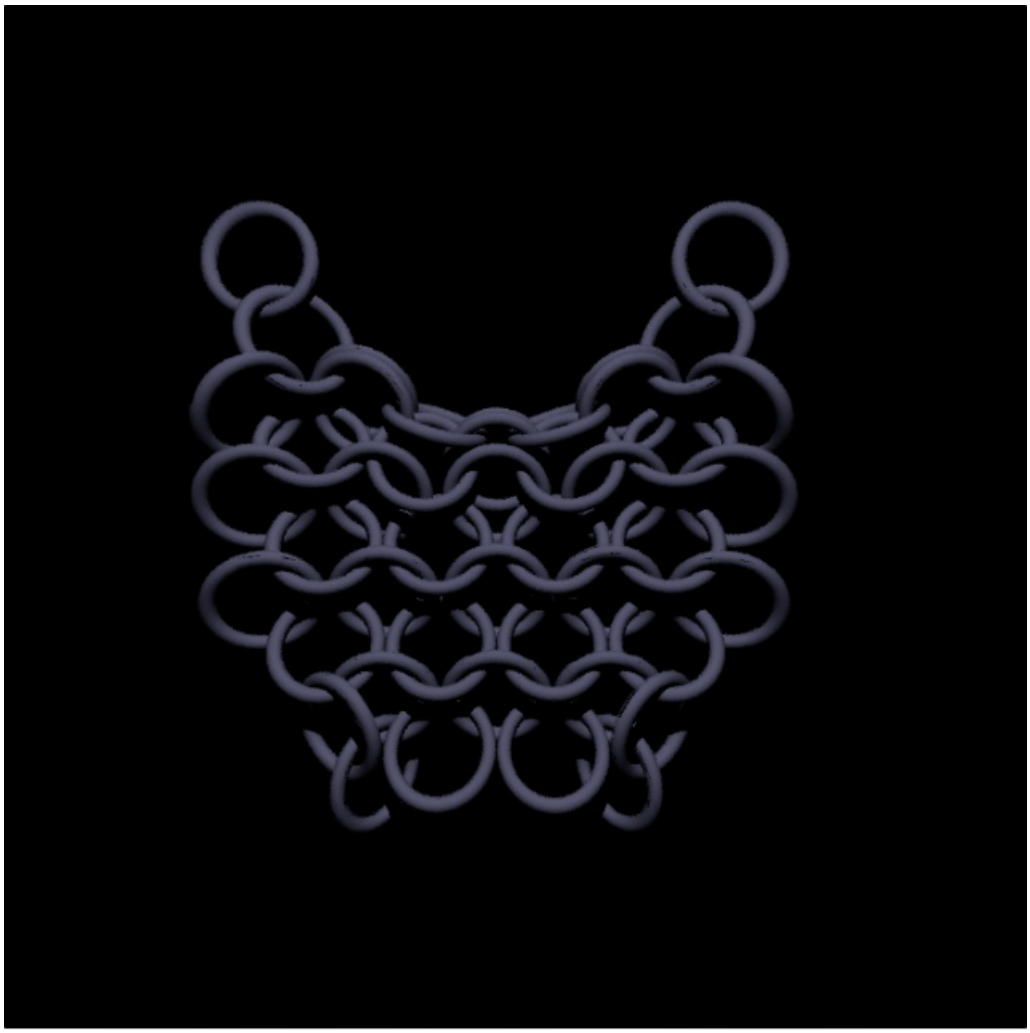

Figure 3: A 5 by 10 piece of 4 in 1 chainmail hanging from 2 corners

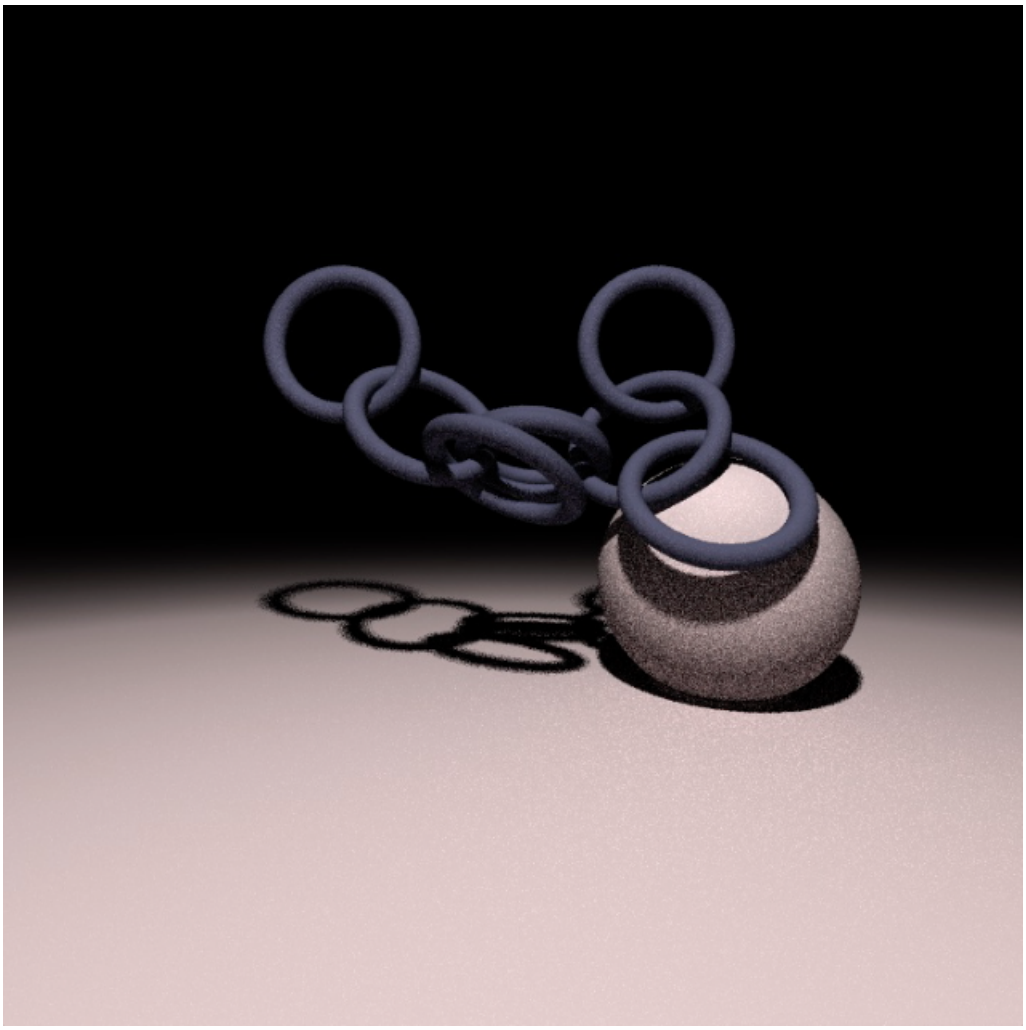

Figure 4: A sphere being rolled into a 3 by 3 piece of 4 in 1 chainmail.