Team Sixteen
Dan Chak
Megan Galbraith
Axel Kilian

6.837 Computer Graphics
Professors: Seth Teller and Fredo Durand
TA's: Addy Ngan and Jingyi Yu

CatenaryCAD: An Architectural Design Tool

Final Project Report

Table of Contents

Abstract

The architect Antonio Gaudi designed complex structures based on catenary systems. His beautiful forms were created by suspending pieces of string from hooks, deforming them with weights and other strings, then inverting the form to create the structural elements.

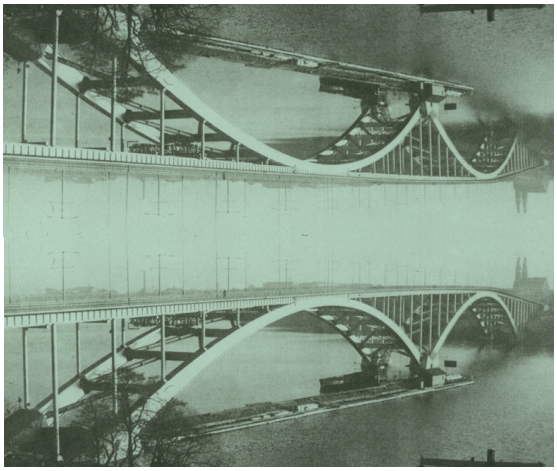
Today's architects are at a loss to reproduce these types of catenary forms when using even the most advanced design tools on the market. For our final project, we aim to provide a computationally enhanced version of Gaudi's atelier. We are creating the design software for architects interested in building models using catenary systems. The tool is implemented in C++ and Tcl/Tk, and intented to be used for both construction and analysis of catenary forms.

Introduction

Antonio Gaudi developed a design technique which allows architects to design complex structures based on catenary systems. The curves in catenary systems are formed by perfectly flexible, uniformly dense strings suspended from their endpoints and weighted under gravity. Gaudi created many amazing structures using pieces of string - structures that architects would be at a loss to try and reproduce today using even most advanced design tools on the market. For this final project, we aim to provide a computationally enhanced version of Gaudi's atelier. Rather than simply allowing an architect to arrange geometric primitives, as they can in AutoCad and other design tools, we instead wanted to provide an environment in which strings responding to gravity and can be arranged to form structures that are far more organic and beautiful.

Catenary systems have been used for construction in Catalan areas of Spain for a long time. For example, if a Catalan stair is to be constructed, the form is not detailed by the planners or architects. Instead, the masons on site hang a rope between the point of departure and the point to be reached, trace the shape, and flip the curve over to use as the guide for constructing the masonry arch that carries the stairs. The rope is in pure tension, as it can not take any compression due to its flexibility. Therefore the form it finds contains the pure tensile force within the envelope of the string. Inverting the parabola results in the pure compression arch necessary for brick construction, which cannot take any tensile forces.

Steel bridge
When inverted, the arches can be identified as catenary shapes (approximately) with the vertical members in pure compression. This bridge is not a design by Gaudi, it illustrates principle of catenary systems.



Antonio Gaudi developed the system of translating catenary string statics into a spatial design system. He constructed scaled models of his design ideas by developing forms through a weighted string form-finding method. In his case, the models are spatial and are much more complex then the catenary staircase example. Gaudi achieved the desired forms

through the control of three variables - anchor points of the strings, the length of the strings, and the weights attached to them. By designing forms this way, Gaudi knew that the resulting geometry would act purely in compression when inverted. He also had a fairly precise estimate of the loads necessary on the different members of his construction. Therefore, Gaudi could construct buildings that would not collapse or require extra support structures.

Beyond structural form finding, Gaudi also used the catenary method for rendering the interior and exterior shapes of buildings. He imagined interiors by painting and tracing over the "wire frame" models of lines, which were simply photos of his string forms.

For this project, we have chosen to create a design tool for architects that differs in its approach to form-finding from current tools likes AutoCad, Rhino, SoftImage, or Maya. CAD packages take the task of drafting and add in the power of computation in order to make more complex and interesting buildings possible. Software tools allow designers to experiment with other shapes, merging computational power with form finding methods that result in more interesting architecture and new styles. Maya and other programs are modelling tools rather than generative tools.

Tools are an essential part of production in any field. Even tool-building itself relies on other tools, with the most basic of all tools being the human hands. People who have the ability to create their own tools are limited only by their imaginations in what they can do. Meanwhile, those that are not tool-builders are limited in what they can accomplish not merely by their imaginations, but also by the tools they have available to use.

In computer graphics, it seems that many of the tools built are meant to service other computer graphics tasks. With the exception of the movie industry, research in computer graphics rarely impacts fields other than the computer graphics field itself. For example, faster rendering techniques and other advances in computer graphics are interesting, but researchers should address how these techniques can be used in new contexts and outside the exclusively visual domain. There are many fields that could benefit from tools that use the state-of-the-art in computer graphics.

Gaudi's rendering technique
Gaudi used photographs of the string models literally as wire-frame models, filling in the surfaces with paint to create an impression of the spaces that would be created.

We hope the creation of our catenary design tool will help contemporary architects realize the beautiful, Gaudi-inspired shapes and means through computational methods rather than physically sitting at their desks tying strings together. Computer-aided catenary designs will be quicker and provide room for playing, trial and error, and potentially provide a means to create more complex designs than imagined in the physical world. In addition, we hope our tool will help expand the reach of computer graphics to outside fields. If successful, perhaps others will begin applying complex computer graphics to new fields as well.

Goals

Generally, we wanted to create a useful and intuitive program for architects to construct precise catenary systems in a three-dimensional world. We decided to set small goals to help realize this grand scheme. One goal was to build was a realistic physical model of the strings so that they would elegantly respond to hooks, weights, and gravity in real-time. We hoped to be able to place the hooks and strings in the three-dimensional world and for them to behave as they would in the real world. We wanted users of our program to have the ability to explore form through simulated gravity-based string modelers. The goal was to provide an easy to use modeling environment solely based on the methodology of placing strings between either fixed hooks or to each other.

Another goal we had for this project involved the visual graphics and rendering component. We were concerned with how the strings, hooks, and weights looked (either realistically or intentionally unrealistically). We hoped that we would be able to have a mode where the strings were rendered in different colors depending on how the parts were affected by external and internal forces, stresses, and strains. We also hoped to have other modes for the user to toggle between when viewing their string constructions, or to have skins that would render over the wire frame structures built up by the strings. We also wanted the user to be able to flip their design 180 degrees and back so they could get a feel for how it would look as a building constructed in the physical world.

Colònia Güell model
Hanging and inverted images of a Gaudi design.

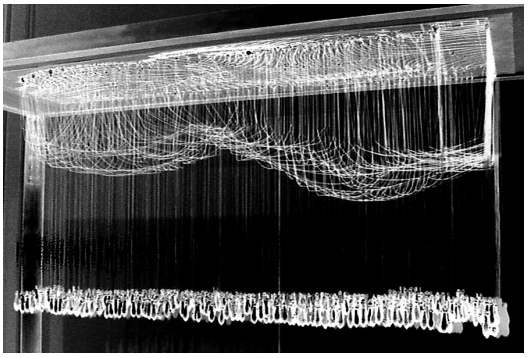
Interiors and exteriors
Sagrada familia still currently under construction.

On the program construction side, we had goals for our data structure design, general back-end program design, and physics computations. We knew that we would need to design intelligent methods for passing information between the different parts of the program, such as a string or hook's placement, length, weight, or the forces applied to them. General information about the design, such as the number of strings in the scene, their adjacency, and connectedness was intended to be available as well.

In order to construct the software for this project within the time constraints, we divided the project into different components, each with small goals we hoped to reach. The first was the construction of an intuitive, clean, and powerful user interface (UI). The UI design was intended to focus on the interactions between the program and the user, in particular with the experience of the user. We wanted to create an intuitive and powerful way to navigate through the program, to allow the user to work within the string-gravity spaces, and to provide the user with the means to save, load, or create new files, and thus new designs. Part of the navigation design meant structuring the layout of the screen and all the elements that interface between human input and the program execution, such as being able to select objects in the scene.

If all went well, we set a goal to user-test our program with some designers in the architecture department at MIT, in order to get feedback on the design of the program, the concept behind the work, and the usefulness of the tool.

Ultimately it was not the goal to just provide a catenary environment but to create a tool that is easily expandable and adjustable. It is also not about recreating an accurate, historical simulation of Gaudi's techniques, but rather to take his inspiring form-finding techniques and use them as a starting point for building a modeling tool that operates around the principles he used.



Weiser- Umemoto model
Explorations of catenary systems in physical and digital models (1996).

Individual Contributions

Each member of the team was responsible for a specific component of the project. We divided the project into the following chunks:

Setting up the development environment
Building the physics simulation/string model
Building the user interface in Tcl/Tk
Writing the code that binds the C++ classes with the UI
Rendering the objects in the scene
Putting together the written report and presentation

Dan Chak's main contribution was setting up the development environment and writing C++ code, in particular the code that glued the user interface to the string model. Setting up the environment involved creating our CVS repository, creating the program framework and writing a makefile, putting all of this into CVS, providing access to the appropriate machines so each team member could work on the project remotely, and offering general system support to the team.

The C++ code that Dan wrote includes the classes cdObject, cdString, cdHook, cdStringHook, cdSkyHook, cdWeight, and cdModel, which create the framework of the program. He also wrote the Tcl wrappers for the C++ class methods, as well as various higher level concepts which were to be invoked from the Tcl/Tk user interface. These functions are in tcl-bindings.C.

Megan Galbraith's main contributions involved designing and building the user interface, designing and formatting the final document and proposal, setting up the team webpage, and building the slides for the presentation. She wrote the Tcl/Tk code that places the buttons and menus in the

windows and on the screen, and made the icons for the user interface. Megan also worked on mapping the correct functionality to the interface objects and setting up the features for users to adjust the parameters of different objects in the model.

Megan was primarily responsible for the visual design of the program, selecting the color scheme, and rendering hooks and other design elements. She worked on ui.tcl, bindings.tcl, tcl-bindings.C and edited other functions in the hook source code.

Axel Kilian's main contribution was to research the best approach to modelling the physics of the strings as they fall in gravity and are deformed by weights, strings, and hooks. He was to then construct a model of this behavior in C++ code. He wrote, adapted, and integrated the particle-spring simulation classes into the project.

Axel built several test series, implemented in Java, early in the development process. They explored the behavior of the spring-particle system in determined and undetermined structural situations. He ran several tests that explored versions of connectivity varying the number of strings that interconnect in a single point. Three strings in a point in space has a unique solution and therefore is still statically determined. If there are four strings there is no one unique solution anymore and one possible solution can only be found through interactive approximation. This is where a system like the spring particle system is necessary. Finite element methods would be perfectly fine to use but pose a much larger workload on the system.

Java Tests
Screenshots from Axel Kilian's three Java physics simulations

In the end, many of these components merged and we each assisted one another with various parts of the work. We all helped to write the OpenGL code for drawing the objects, we worked together to debug the program, and chipped in with the writing of the final documents.

Achievements

All in all, we achieved the spirit of the program we wanted to make, although the functionality is not as far along as we'd hoped it would be. There were setbacks due to the languages that we used to code the project because two of the three team members had little to no experience using either Tcl/Tk, C++, or both. Also, the particle system we used to model the strings was problematic in the C++ version, and it kept us from adding additional features and modes to the tool that are more interesting than the basic behavior we were able to implement.

In terms of being able to interact with and use the system, we achieved most of our goals, although in two separate paths that the program took. In one path, many of the manipulation functions for accurately placing objects manually and for developing 3D models functioned, but the solver was highly unpredictable. The strings performed what looked like a laser show across the screen. These problems related to ill-behaved physics, so we stepped back and rewrote parts of the program. The new iteration resulted in a program where the solver behaved well, but the 3D placement and manipulation was non-functional. We were unable to get the two paths working together as we had set goals to do.

The user interface as a whole contains the important methods and functions for placing strings and hooks into the world so that designs can be built. The canvas of the scene is interactive, and users can point and click to place objects in the space or to select objects, which we do by comparing distances to the mouse position in the x-y plane. Selecting objects in a 3D world was a more difficult task than we expected. However, it is worthwhile because the user can get feedback from the program about the position of hooks, lengths of strings, and placement of hooks on strings.

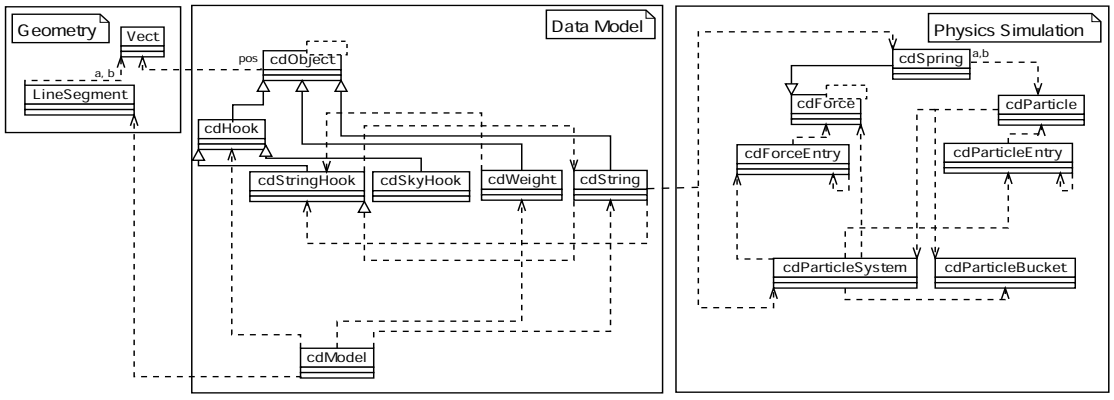
We made small achievements with the physics modeling problem and getting the solvers and particle methods to behave reasonably. The strings did not always behave as they should have, but there are elements about their behavior when working properly that really add to the organic feeling of the tool. Most notably is when new strings are created and fall with respect to gravity. To work on this component, Axel created four Java versions of what the particle system should do under the deformations of hooks, weights, and strings. These versions were difficult to successfully realize in the C++ interactive version of the program. However, the conceptual work was done and as a result we have several very nice applets that show this in concept.

We reached the goal of providing the necessary infrastructure for the program and setting up the interactivity. We also reached the goal of creating a fully working prototype that allows for all the fundamental modes of modeling the catenary systems. The main hurdle was the combination of interactive user input with the constantly changing spring particle model.

Finally, it was an achievement to set up a development environment for three programmers with different backgrounds to work together on a single project. We were concerned about getting in each other's way or overwriting code, but wanted to be able to work on the project without

needing too much interaction with the entire group to merge program pieces and keep everyone's code in sync. Using CVS was a wise choice for us, particularly when we would break things that previously worked and we wanted to step back to older versions to see what went wrong. Dan was successful in building the interface between the C++ code and the Tcl/Tk code so that Megan and Axel did not need to concern themselves with the behavior of each others component when writing their code for their parts.

The following is a description of the system we built for the CADenary design program.



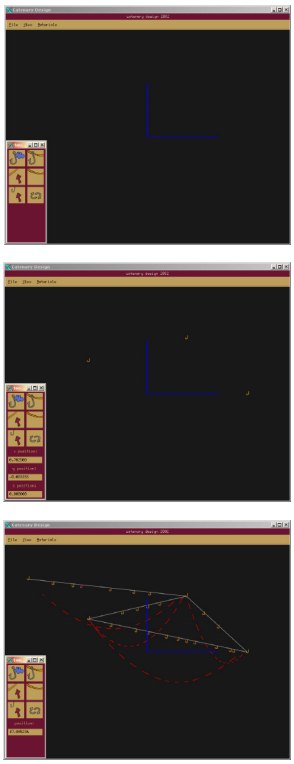
Program Data Model

CADenary's data model consists of hooks, strings, and weights. There are two types of hooks, SkyHooks and StringHooks. A SkyHook can be placed anywhere in the work area, whereas a StringHook hangs on a string. Strings can hang between any two hooks in the work area. Weights can be attached to StringHooks that are first attached to strings.

SkyHooks, StringHooks, Strings, and Weights are all implemented as C++ classes. Another C++ class, Model, is a container class for all of these objects, and represents the entire work area. This can be seen in the Object Model Diagram above in the box labeled "Data Model."

Some functions were needed in order to build the catenary model and use the program well. For instance, we needed to be able to place hooks both in the sky and on strings. We needed to be able to select strings and hooks by clicking on them so that parameters could be retrieved or strings could be attached. We also needed to look at the model in several modes, by viewing it in the x-y plane with the correct orientation, by viewing it in the x-y plane but turned 180 degrees, or by panning around it in the three-dimensional space.

We implemented a method to find the closest object to the mouse click. Whenever the mouse moves, a function is called that finds the closest



Debugging
Screenshots from a version with a misbehaving physics simulation.

object to the mouse pointer for each type of object. For hooks and weights, this is trivial. The distance from the mouse pointer to each hook or weight is calculated, and the smallest distance is chosen. Strings are slightly more complicated. The distance from the mouse pointer to each segment of each string must be calculated. Again, the string which is the smallest distance away is chosen as the active object.

In addition to navigation and the usability of the program, we needed to construct a model for representing the strings so they behave realistically in real time. The strings are placed as segments around the space, and each segment is calculated either from hook to hook, as in the Data Model representation, or from particle to particle, as in the Physics Model representation. These models do not necessarily correspond or behave similarly, and we have struggled to decide the best way to relate these two very related representations. One possibility is to abandon the Data Model representation of strings (connections from hook to hook) and use the Physics Model representation for both computation and user interaction. Another possibility is to make each StringHook correspond to a particle in the Physics Model. This is what we are attempting to do.

Tcl/Tk User Interface

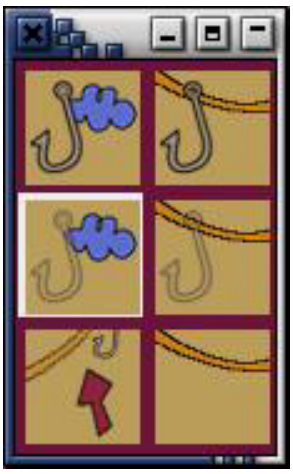
The Tcl/Tk user interface sits on top of the C++ Data Model. This part of the program represents the menu bar, canvas, and the toolbox. All the classes represented by the Data Model have Tcl function wrappers which are called by the Tcl program in an event-driven manner based on the user's actions. The Tcl program passes data to the C++ program when the user selects a tool or menu option, clicks in the work space, or simply moves the mouse around the screen.

The Tcl/Tk user interface provides several options to the user:

Create a new SkyHook
Create a new StringHook
Create a new String
Select an object
Retrieve/modify the parameters of an object
Rotate the scene 180 degrees to view structures
Real time graphics window
Interactivity

Most of these options can be accessed through either the menubar across the top of the main window, or by click on the corresponding button in the floating toolbox. This sets the program into the correct mode and lets

UI Toolbox
Screenshots of an early version of the user interface toolbox. Buttons let users place SkyHooks, StringHooks, and Strings, among other things.



the user perform their duties until a new option is chosen. For instance, if the user is interested in setting the parameters of an object by hand rather than relying on the inaccuracies of the mouse, they may select the object then modify the parameters in the toolbox window.

Each object has certain parameters than can be retrieved by the user and potentially modified. Selecting or placing a SkyHook lets the user see the x, y, or z coordinate of the hook in space, whereas selecting or placing a StringHook lets the user know how far down a string the hook is placed. If the user selects or places a String, the Tcl/Tk interface informs the user of the length of the string. Initially, the length is one and a half times the distance between the two hooks. Each string is divided into thirty particles that have individual forces acting upon them, and StringHooks can be placed along these particles.

All icons used for the toolbox buttons are originals. They were made for this project using Adobe Photoshop. The colors in the background and menus are neutral, earthy tones in order to be pleasing to the user's eyes and to reflect the organic properties of the strings used to build the catenary models. The renderings of the hooks and strings were kept very minimal for several reasons. First, we agreed that a simple, elegant interface would make the tool more effective. Second, we wanted to ensure that the program ran smoothly in real-time without being bogged down with complicated renderings that distract from the form in construction.

Physics Simulation

We chose a Spring-Particle system for simulating the physics-based string behavior. This choice was based on our need for a robust, interactive simulation method that would allow us to interact with the string model as it is simulated. The Spring-Particle model in itself does not guarantee robustness - its behavior for larger numbers of random combinations of springs and particle chains depends in a large part on the solver used in the system. We are using the Runge-Kutta solver, an explicit solver. It produces reasonably robust solutions for most cases.

It is susceptible to breaking when the scale of strings change or excessive stress is introduced into the springs through stretching the string. Another shortcoming of the current Spring-Particle System is the lack of conservation of length of the string. The system becomes very unstable if the stiffness parameter of the spring is set beyond 1. At the highest stable setting the string experiences approximately a stretch factor of 1.5 times of the at rest length which is far more then a common physical string would exhibit. Currently we do not place emphasis on the conservation of the length of a string but rather interact with the system based on visual feedback. But this problem has to be addressed in further iterations of the project.

The Particle Spring model is implemented with the use of the following classes. The particle functionality is handled in cdParticle.C, cdParticleBucket.C, and cdParticleEntry.C. The Spring implementations occur in cdForce.C, cdSpring.C, and cdForceEntry.C. The cdParticleSystem.C class controls how forces and Particles relate to each other. In order to create a string, the classes cdParticle.C and cdSpring.C are instantiated to form a chain of Particles, interconnected by springs.

The parameters of a Spring are the rest length, the length when the spring is not in tension, and the damping factor, which is at 0 in our case since it has the tendency to spin the model out of control. Particles have a mass parameter, a location, and know whether they are confined or free to move about. The end hooks of strings and the end particles are fixed to the hook positions. A special case is when a string is attached to a StringHook. StringHooks follow the movement of the strings they are attached to. cdParticleSystem.C deals with the calls that update the ParticleSystem. It invokes a Runge-Kutta solver to solve for all the forces involved in the update cycle.

The program structure can be seen in the Object Model Diagram in the box labeled "Physics Simulation."

Lessons Learned

One of our original motivations for this project was the statement that designers are limited by their tools. If your CAD program can only draw right angles, you are going to make a lot of rectangular structures. What we thought in the beginning couldn't have been more true. Not only was it easy to quickly figure out what you could do with CADenary to make a structure, but it was also easy to figure out what you couldn't do. Even if a certain type of interaction would get you in trouble because of the physics simulation, this sort of limitation would present itself plainly and you could design right around it. In about twenty minutes, Dan created a structure that looked a lot like the Eiffel tower (but would be self supporting!) using CADenary, even in it's extremely nascent stages.

While C++ might be the best language to use in actual development of a computationally intensive, complex program that uses 3D graphics, it may not have been the right choice for proof-of-concept work such as what we were aiming to accomplish for this class project. Swing and other Java APIs can be just as difficult to pick up as Tcl/Tk or the C++ STL, but Java may have been a better choice in the end because of the experience of the people in the group. Many features in the user interface and the

construction of the physics model would have been built much quicker and behaved better had they been done in Java. Much of our lost time was a direct result of Axel needing to grow more comfortable writing code in C++, and Megan needing to learn Tcl/Tk. Axel and Megan both learned about developing complex program structures in a collaborative setting in C++ and Tcl/Tk, which they did not have prior experience in.

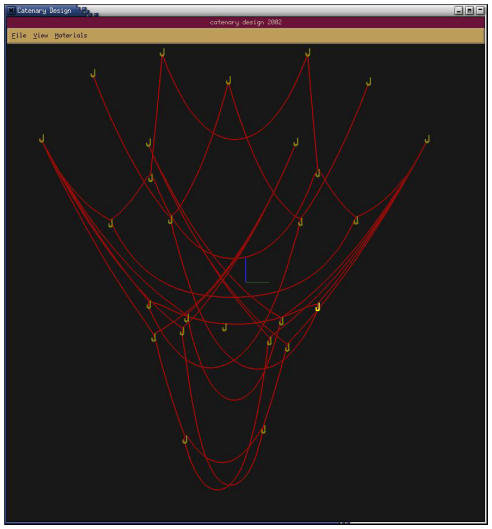
Using one unified language for all aspects of the program (rather than the Tcl/Tk and C++ split we chose) may have made development smoother as well as more rapid. One of the original reasons for choosing C++ was to have access to the openNurbs library so that we could build skins, surfaces, and save the models for use in 3D printers . This was never realized simply because we didn't make it that far.

Another lesson we learned was that it would have been more effective if we'd been able to develop the code on a variety of platforms rather than only Linux.

Investigating the pros and cons of physical simulation engines was an interesting task because we had not previously looked in depth into this field. One lesson learned was the importance of choosing the appropriate solver for the task at hand based on speed requirements, robustness and required precision of the calculations. An important part of simulations is to chose the appropriate criteria in order for the results to be meaningful within the chosen setting.

Deliverables

We are able to provide the following deliverables to the class upon the completion of this project.



A working CADenary program with basic features
Three Java test applets illustrating physics concepts
A collection of short movies
Source code for the project
Final Project Paper online in PDF form
Final Presentation

Acknowledgements

We would like to acknowledge the following people for their help and support while we worked on this project, whether they knew they were providing it or not.

Prof. John Ochsendorf, Axel's structure professor in Building Technology, for meeting with him to talk conceptually about the project, in particular about potential problems with solvers.

The 6.837 teaching staff, in particular Fredo Durand, who met with Axel, Seth Teller, for sending an encouraging email when his colleague showed interest in the project, and Addy Ngan for meeting with us once a week, for admitting he didn't have a clue what we were talking about when we proposed our idea, and for not getting too discouraged when we didn't stick to our timeline of progress.

Cat Foo for letting us invade her home when the conference room down the hall was full, for making fresh soymilk for Megan in the mornings, and for providing Dan and Megan with comic relief during the semester.

The Aesthetics + Computation Group, in particular Simon Greenwold, Tom White, and Ben Fry, for their little snippets of knowledge and Prof. John Maeda, who trembled when he called us the "Power Team."

Nicola Stafford, who didn't get mad when Megan turned down tickets for Guns 'N Roses so she could attend a meeting with her group.

Lor sim augait,se tat et, quis accummy nulla facin ut autat lutpat adigna faccum quam qui tem dit del et ut laoreetum dolore tat. Ut landiam, sequis aciduis acip et lut vulputpat.

Andrew Boardman and the Athena hotline for getting us music on the IBM linux machines in the building 66 cluster (but not Seth Gilbert, who wouldn't let us install Athena-Linux in the Sidney-Pacific cluster. Grr).

Bibliography

Faure, Francois. Fast iterative refinement of articulated solid dynamics. To appear in IEEE TVCG, 1999. http://w3imagis.imag.fr/Membres/Francois.Faure/

Jakobsen, Thomas. Advanced Character Physics. http://www.ioi.dk/Homepages/thomasj/publications/gdc2001.htm. IO Interactive, Denmark. 2001.

Meltzer, Jonathan D. Gaudi Central. http://www.op.net/~jmeltzer/Gaudi/colonias.html. March 18, 1998.

Ottos, Frei. Institute for Lightweight Structure and Construction, ILEK. http://www.uni-stuttgart.de/ilek/Fotoarchiv/Fotoarchiv.html

Smith, Jeffrey, Hodgins, Jessica K., Oppenheim, Irving, Witkin, Andrew. Creating Models of Truss Structures With Optimization, Proceedings of Siggraph 2002. ACM Press, New York. 2002. pp. 295-301.

Solé, Eduard. The Crypt of the C hurch of Colònia Güell (1898-1916). http://www.gaudiclub.com/ingles/i_vida/colonia.html. 2002.

Weisstein, Eric. Catenary. World of Mathematics. http://mathworld.wolfram.com/Catenary.html. CRC Press LLC. Wolfram Research, Inc. 1999.