RICE UNIVERSITY SCHOLARS PROGRAM
COVER SHEET FOR PROPOSAL

PROJECT TITLE	Geometry Synthesis	
NAME	Mathias Ricken	DATE 11/04/03
RICE ID	0812203	MAILING ADDRESS
E-MAIL	mgricken@rice.edu	6360 Main St.
PHONE	(713) 348-1940	Houston, TX, 77005
ADVISOR	Dr. Joe Warren	
DEPT.	Computer Science	
PHONE	(713) 348-5728	TOTAL FUNDS REQUESTED:
E-MAIL	jwarren@rice.edu	$1400

ABSTRACT OF PROPOSED STUDY

The creation of meaningful geometry for architecture or simulations currently takes a considerable amount of time. Traditional geometry generation usually entails the creation of a rough layout of the architecture, the manual addition of geometric detail, and a final application of colors and textures to the geometry.

By automating large portions of this process using the parametric generation of large-scale geometry and the subsequent addition of detail using techniques borrowed from texture synthesis, geometry generation will be accelerated dramatically.

I propose to study algorithms that allow for such automatic geometry synthesis and to create an editing environment that enables the user to automatically generate architecture governed by a set of parameters. In a second phase, the environment will then be used to synthesize meaningful detail into the still bare geometry.

The system will display the results 3-dimensionally and let the user export the generated geometry into other programs.

TABLE OF CONTENTS

INTRODUCTION

Computer-generated, 3-dimensional geometry is ubiquitous today. It has been used for over a decade in *computer-aided design* (CAD) of commercial products and the simulation of architecture, is a major component of current computer games, and has been embraced by the movie industry to cheaply replace reality or to build an alternate one. Education and training also make increasing use of 3-dimensional graphics: Simulator training has become an integral part of a flight school's curriculum, and NASA practice missions are unthinkable without the use of computer simulations. Medical schools and the United States army have also begun to use 3-dimensional computer graphics in their training programs.

The creation of meaningful geometry for these architectures and simulations, however, currently takes a considerable amount of time. Traditional geometry generation usually entails the creation of a rough layout of the architecture, the manual addition of geometric detail, and a final application of colors and textures to the geometry. In the computer game industry, for example, it requires roughly the work of an entire man-year to create the content for only 40 hours of game play [1].

Intelligent algorithms can reduce this figure significantly and open up new opportunities. The content of a simulation project can be larger and more realistic, while the costs of creating the project can be reduced. By automating large portions of the traditional procedure described above, the designers have more time to tailor the simulation to their specific needs. In computer gaming and training, fast automatic generation of new environments also increases replay value and the effectiveness of the simulation.

Aside from the immediate practical benefits, the successful completion of this study would also tie together two research areas that are currently disjoint: textures and geometry. A success would prove that texture synthesis techniques can be applied to

geometry as well. Using this connection to leverage texture synthesis techniques would be immensely useful and increase the value of both previous and future work these areas.

A connection between textures and geometry might also allow us to employ optimizations in texture synthesis that have thus far only been applied to geometry. To reduce the amount of information, for example, geometry is often represented as an *octree*, a data structure that divides space into eight quadrants. If one of those quadrants is completely filled with the same material, the quadrant does not have to be broken down to the finest resolution but can instead be represented as a larger block. Using an octree representation for synthesis by analogy might reduce the number of comparisons necessary to find the best-matching neighborhood and thus result in improved runtimes for both texture and geometry synthesis.

PREVIOUS WORK

Previous work that falls precisely into this area of study, the generation of meaningful geometry on an architectural level, is very limited. Most work has been done in fields that operate on a different scale; the geometry created either covers a large area but lacks detail, or is rich in detail but constrained to a small space. Work that deals with the level of detail my proposed research is concerned with has thus far suffered from many restrictions. Texture synthesis research is not inherently related to geometry, but serves as source of techniques that can be applied to geometry synthesis. This section will examine work in all three categories.

Planetary-Scale Geometry Synthesis

Most work regarding geometry generation has been done on a landscape basis, and several software products are available already [2][3]. The algorithms at the cores of these programs mostly utilize a noise source, such as Perlin noise [4], to generate a matrix called ***height field***. Small numbers in this field correspond to low altitudes, and large numbers to high altitudes. The user can then apply filters to the terrain that smoothen, roughen, or erode the ground, change the sea level, or add rivers and lakes. Using these tools, mountain ranges, lakes, entire continents and planets can be generated and photo-realistically rendered, as demonstrated by Terragen in Figure 1.

Some systems even include a simulation of a plant ecosystem [5]. Cellular automata, as found in the ***game of life*** [6], for example, determine the distribution of the different plant species. Certain plants can only survive in well-irrigated areas, while others are more resistant to draughts. Plant seeds of one kind may spread more easily than those of another. All these parameters determine the survival, breeding patterns, and ultimately the placement of plants. Again, the results are excellent, require very little human interaction, and nearly photo-realistic (see Figure 2).

These methods of generating geometry function on a completely different scale than the ones I am proposing. The algorithms create entire continents and planets; they simulate the constructive forces of nature, not those of man. Even though an ecosystem similar to the one used for plants could possibly simulate human settlements, the

Figure 1: Terragen Landscape

Figure 2: Plant Ecosystem

simulation would again happen on a very large scale. The simulation of an ecosystem cannot be applied to only a single building. Architecture, that is construction with a coherent form as a result of a conscious act, seems inherently different from the way nature shapes a planet; the generation of architecture, therefore, cannot be simulated using the same means.

A combination of the two approaches should be possible, though. Such a system would use noise- and filter-based geometry synthesis on a large scale to create a natural environment. The graph- and analogy-based algorithms I propose then fill this habitat with structures. Merging the two types of algorithms, however, is a subject that will require further investigation.

Maze Generation

Some previous works document algorithms for creating smaller structures [7]. One such algorithm, recursive backtracking, involves randomly removing walls from a grid and moving into that direction if that cell has not previously been reached. Once the algorithm encounters a dead end, that is a cell where no wall can be torn down since all of the surrounding cells have already been explored, the program backtracks to the closest cell where a wall can be removed. Applying this procedure until the entire grid has been processed results in a Mephistophelean maze, depicted in Figure 3. Since every place in the maze can be reached in one and only one way, these kinds of mazes are also called perfect mazes.

While Mephistophelean mazes have nice properties, most of the time these labyrinths do not reflect human architecture. The mazes are too repetitive, restrictive, and use space in an awkward way. Additional shortcomings are that the algorithms generate mazes on some kind of regular grid, and that they are mostly 2-dimensional in nature.

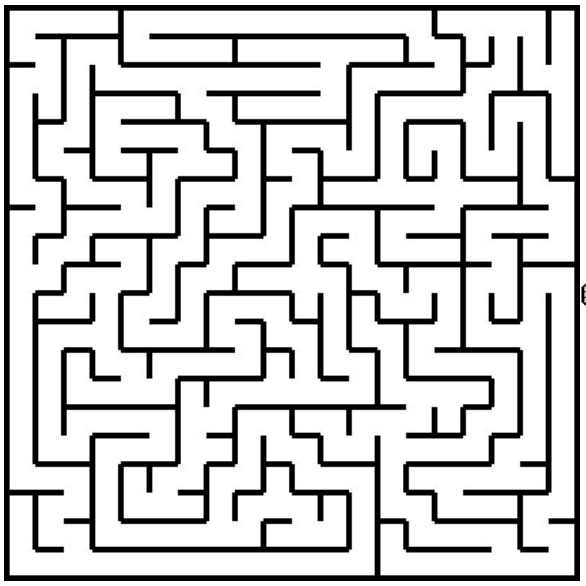

Figure 3: Mephistophelean Maze

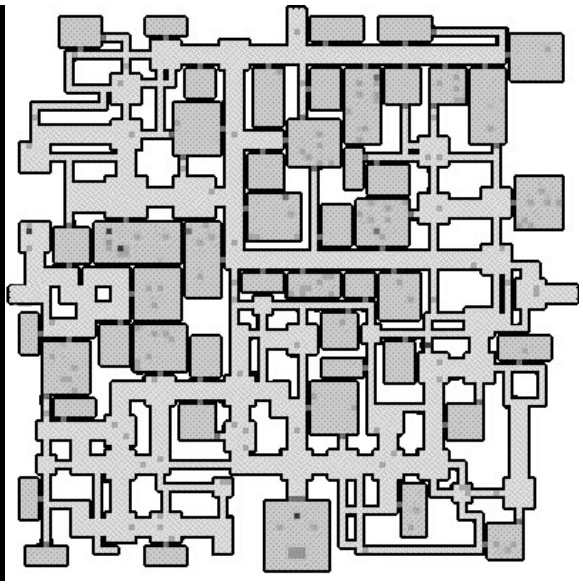

Figure 4: DungeonMaker Maze

The DungeonMaker program employs a different strategy [8]. The user specifies starting locations for agents called *tunnelers*, which excavate corridors and change direction according to a probability table. At random places, tunnelers can spawn new agents that branch off in different directions, or create rooms that are sectioned off from the corridor by doors. Before generation, the user can also manually place corridors, walls, and doors.

If care is taken, the resulting maze still exhibits connectedness, but allows loops, wider tunnels, and generally more interesting shapes. Mazes generated by this tunneling algorithm look more realistic. They contain corridors, halls, and rooms reminiscent of human architecture, as shown in Figure 4.

Currently, the algorithm still shares some of the disadvantages of Mephistophelean mazes: The procedure operates on rectangular grids and is exclusively 2-dimensional in nature. I am certain, though, that the tunneling algorithm can be modified to make free use of space in three dimensions.

Texture Synthesis

Much more research has been done in an area not directly related to geometry synthesis. In texture synthesis, an algorithm creates new textures or images from one or more samples given to the procedure as input. The image should then be similar to the originals but not be exactly the same.

In his doctoral thesis, Li-Yi Wei described an algorithm that is often used in texture synthesis [9]. His technique requires a source and a template texture as inputs and generates a synthesized texture of arbitrary size that resembles the source texture (see Figure 5). In most cases of regular texture synthesis, the template texture just contains random noise, even though other possibilities exist, as I will explain later.

The procedure cycles through every pixel of the template texture, beginning with the top left pixel, moving left to right, top to bottom, just like text is normally read in English. The program looks at a pixel's *neighborhood*, the set of pixels that surrounds it, and compares the neighborhood in the template texture to all possible neighborhoods in the source, and then selects the one that is the least different. Usually, an L_2 norm is used, which compares the sum of the squares of the differences. Since the differences are squared, the norm punishes large aberrations more harshly than small ones, resulting in closer matches. Once the procedure has found the best neighborhood in the source texture, the algorithm copies the pixel in the center of that neighborhood into the current pixel in the template and continues with the next pixel.

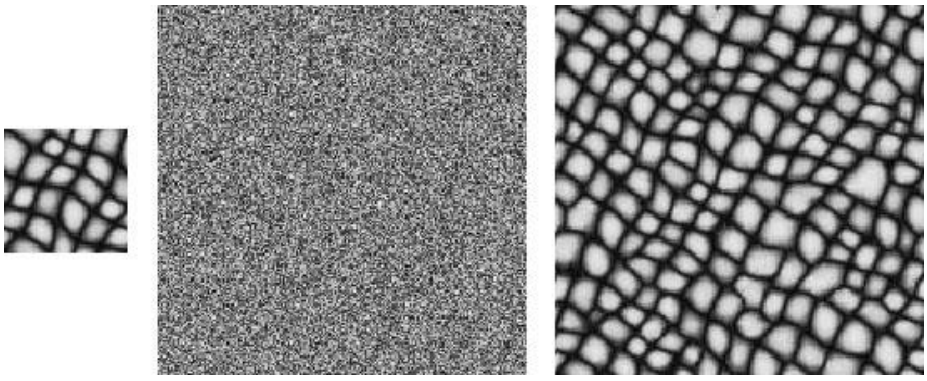

Figure 5: Texture Synthesis (left: source texture; middle: noise; right: synthesized texture)

Neighborhoods vary in size and shape. Larger neighborhoods can reproduce larger features in the source texture; therefore, the size of the neighborhood should be proportional to the size of the features in the source. The proportionality constant is called the *randomness parameter*; small values lead to textures that hardly resemble the source, while values close to 1 or larger recreate the source almost perfectly.

If the template contains random noise, the neighborhood also needs to be *causal*, that means the synthesis process should only take those areas of the texture into consideration that have already been synthesized. Because of the way pixels are processed, the areas above and to the left of the current pixel have already been generated. A causal neighborhood therefore generally has the shape of an "L" rotated 90 degrees clockwise. The requirement for causality thus determines the shape of the neighborhood.

The template, however, does not have to contain noise. The texture might start off with meaningful information or a mixture of both meaningful information and noise. In this case, the algorithm will attempt a *constrained texture synthesis.* The process is constrained since the synthesized pixels must blend in perfectly with the ones already present, some of which can be in areas not already processed located below or to the right of the current pixel. Therefore, a symmetric neighborhood is necessary. If the image contains noise, a two-pass algorithm maintains causality: The first pass uses an L-shaped neighborhood; a second pass with a symmetric neighborhood satisfies the boundary requirements. Constrained synthesis can be used to restore missing pieces in a texture, to erase certain areas, or to expand the texture.

As stated above, larger neighborhoods are necessary for the reproduction of large features. Unfortunately, running time increases dramatically with the size of the neighborhood. The algorithm depends on the size of the template texture t, the size of the source texture s, and the size of the neighborhood n. Since the program compares every neighborhood in the template with every possible neighborhood in the source, and every pixel in a neighborhood with the corresponding pixel in the source neighborhood, a naïve

implementation is $O(t^2s^2n^2)$; doubling the size of the neighborhood quadruples the running time.

Several optimizations have been proposed to reduce running time. In his thesis, Wei already implemented a multi-resolution algorithm that uses a pyramid scheme to recreate large features while using smaller neighborhoods. Both the source and the template texture are halved in size, and synthesis begins at a much lower resolution. Since the size of the largest feature is also halved, a smaller neighborhood is now able to capture the feature. The algorithm then doubles the size of the results from the low-resolution synthesis again, and the procedure continues at the original resolution with the rescaled results as template.

Together with Marc Levoy, Wei also described how to accelerate texture synthesis by using tree-structured vector quantization (TSVQ), a common technique for data compression [10]. Instead of doing a linear search through all possible neighborhoods, the modified algorithm constructs a tree of neighborhoods before synthesis begins. Search time is therefore reduced from $O(n)$ to $O(\log n)$, where n is the number of neighborhoods that are compared.

Lin Liang et al noted that most of the time, large contiguous patches are transferred from the source texture into the template [11]. Best-match comparisons only need to be made at a patch's border where it blends into other patches. In patch-based texture synthesis, the algorithm breaks the source down into individual patches and pastes them into the template in their entirety. When the procedure selects a new patch, this method only compares the borders of the patches, which reduces the number of comparisons and greatly improves the running time.

While texture synthesis does not directly relate to the task at hand, it is possible to change the representation of closed polygonal geometry so that the techniques described above can be used. The space containing the geometry is divided into a uniform grid, and the distance of a cell to the closest polygon is calculated; the distance is positive or

negative depending on whether the cell is inside or outside the geometry. The result is a matrix of numbers, not very different from the one used to represent a texture. The synthesized matrix can then be transformed back to polygonal geometry using *Marching Cubes* [12] or *Dual Contouring* [13].

During the summer of 2002, I have already demonstrated that texture synthesis techniques can be used to generate new geometry. Figure 6 shows hand-created geometry on the left and the result of geometry synthesis on the right.

Unfortunately, my studies showed that the algorithms perform poorly when attempting to reproduce large-scale structures. This was somewhat expected, since

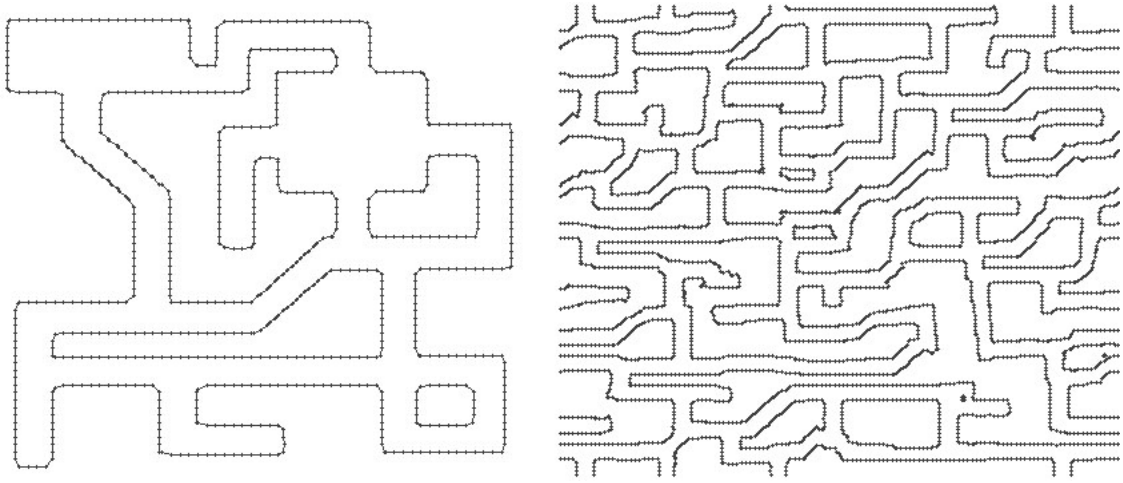

Figure 6: Geometry synthesis by analogy (left: source; right: synthesized)

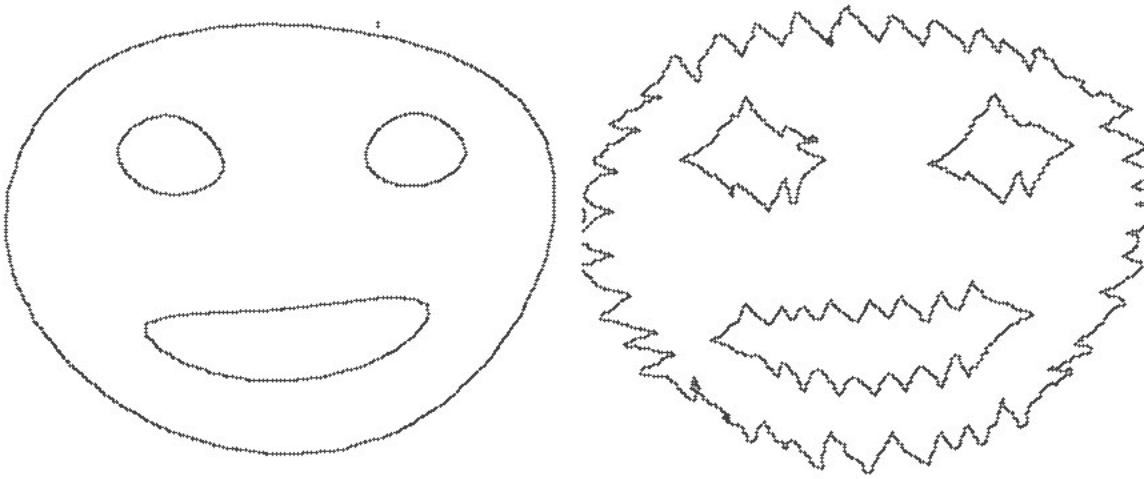

Figure 7: Surface properties modified by synthesis (left: source; right: synthesized)

texture synthesis techniques require the image to be stationary and local, as explained in Wei's thesis; applying the techniques to large-scale geometry thus fails for the same reasons the algorithms originally fail to produce *pictures* as opposed to just textures. Attempts to use synthesis by analogy to modify surface properties yielded some encouraging results, though, as Figure 7 depicts. I would like to pursue this direction further.

PROPOSED WORK

I propose to build an editing system to synthesize 3-dimensional architectural geometry. The system will consist of two editing stages and allow user interaction or automatic generation at any time. The program will also have a graphical user interface (GUI) and show the results of the synthesis in a 3-dimensional display.

The first stage will construct large-scale architecture using a parametric algorithm similar to the one DungeonMaker uses. I will have to generalize the tunneling algorithm to work with 3-dimensional, not grid-based spaces. This process also involves defining precise parameters to control the generation of geometry, such as the frequency that a path branches, the possible angles for those branches, the thickness of the walls, alignment options to create grid-like layouts and floors, the sizes of corridors and rooms and their shapes. Since the original tunneling algorithm performed well on a 2-dimensional uniform grid, I expect good results from the extension of this algorithm. I believe it will be hard to find the parameters that will lead to architecture as we know it today, though. Creating architecture that actually resembles today's buildings will need further investigation, as architecture is always connected to a purpose, which is beyond the understanding of the program. The synthesized geometry should nonetheless be interesting, diverse, and suitable for use in a computer game or simulation, for example.

Should the proposed algorithm unexpectedly perform poorly in the modified environment, I will implement an alternative algorithm that has two stages. The first

stage places points according to some of the parameters mentioned above. Depending on how close the points are, the procedure will connect some of them to form a graph. In the second phase, the algorithm then picks suitable geometric objects to transform the points into rooms and the connections into corridors. My provisional investigations suggest that this procedure will work as well; however, the tunneling algorithm described above has already proven to yield good results on a 2-dimensional grid and therefore seems more promising. I will also be able to implement the tunneling algorithm more quickly than the untested algorithm, giving me more time to focus on geometry synthesis by analogy.

A successful parametric algorithm produces a set of rooms and corridors similar to human architecture. The geometry will follow the parameters specified above and interconnect with tunnels that have been placed manually. All rooms will be reachable unless the user expressly desires the opposite.

Once the raw geometry has been created, algorithms borrowed from texture synthesis will automatically add detail to the architecture. The user will be able to select an area of the geometry and different detail sources, and the program will apply those details to the area. The changes made can be both geometric and textural in nature; they might change the geometry to match that given in the sources or automatically add seamlessly tiling textures to the selected area.

Figure 8: Geometry Synthesis by Analogy (left: raw geometry; middle: source; right: synthesized)

Geometric changes to the architecture might, for example, involve adding a brick pattern with actual seams to a wall, installing windows in an outside wall that match the height and style of the wall, adding steps and a railing to convert a ramp to a staircase, and possibly augmenting a room by including furniture, plants, and other objects. Figure 8 provides an illustration of how the algorithm might add windows to a wall. The textural modifications will be more traditional and follow the classic ideas of texture synthesis to automatically paint the geometry in a coherent fashion.

To implement geometry synthesis by analogy, I will have to generalize the synthesis algorithms I have implemented in 2002 to work with 3-dimensional geometry. Transitioning from two to three dimensions will degrade runtimes even further and demand that I make use of the optimizations the works in texture synthesis describe. I expect particularly good results from using patch-based synthesis since it reduces the number of comparisons to a fraction of the original count. I cannot make any statements about the time the editing system will require to synthesize a level because I have not worked with these particular optimizations; the unoptimized synthesis of 2-dimensional geometry sometimes took several hours, though. The time to generate the raw geometry will be much shorter and the program will need no user interaction while the program is synthesizing, so using this system will still be beneficial to designers.

Successfully implementing synthesis by analogy will allow the user to automatically create geometry and texture patterns that match the styles selected as inputs. The patterns will contain few visual flaws and be seamless unless sharp features are desired. The quality of the match can be measured by analyzing the ***energy*** of the result, the sum of the differences between the environments in the template and the best matches found in the sources. A good synthesis result will have a low energy and thus conform more closely to the source geometry.

There is no alternative plan if geometry synthesis by analogy fails to provide good results. While I will still be able to demonstrate the generation of raw geometry and the

automatic texturing of the architecture without geometry synthesis by analogy, my main interest is to prove that algorithms for texture synthesis can be used with geometry as well. Due to the successful demonstration of geometry synthesis in two dimensions on a small scale and the promising results in changing surface properties, though, I am certain the algorithms can be successfully leveraged.

Several approaches exist to improve the synthesis result. In my previous studies, I noticed that the application of several passes over the same area generally produces results that match the source geometry better. The algorithm could be modified to run several passes in areas where the energy of the synthesis remains above some threshold value.

Encouraging the selection of pixels from adjacent neighborhoods is another possibility. If the last pixel was generated from one source neighborhood, the area surrounding this neighborhood could be given a higher probability of being picked. This technique leads to larger patches selected from the same source region and therefore to synthesized geometry that is more contiguous.

The final results of the synthesis process can either be viewed in the editing system or exported into other programs. The exporting capability will make using the geometry in other programs possible, allowing further enhancements.

Budget

For the completion of this project, I request funds for the items below. I would also like to demonstrate my results at the ACM SIGGRAPH 2004 conference held from August 8-12, 2004 in Los Angeles and thus request funding for travel and registration expenses.

Unfortunately, my work will not have progressed enough to allow me to submit a paper; however, submission of a poster and participation in the ACM Student Research Competition is possible. Being able to showcase my research at SIGGRAPH and to explain its implications personally would be of great value. Registration in advance is possible.

Sources	Copying and printing; access fees to papers; books (to be donated to Fondren Library)	$175
SIGGRAPH	Conference registration	$425
	Travel	$500
	Lodging (4 nights)	$300
Total		**$1400**

Timeline

The anticipated timeline for this project is as follows:

December 2003

Large-scale geometry generation

February 2004

Proof of concept for geometry synthesis by analogies

April 2004

Optimizations for geometry synthesis and thesis

May 19, 2004

Submission of poster to SIGGRAPH 2004

August 8-12, 2004

Presentation of work at SIGGRAPH 2004

AUTHOR'S QUALIFICATION

During the summer of 2002, I conducted initial research on geometry synthesis under the supervision of Dr. Warren. In the beginning of these studies, I successfully implemented several texture synthesis techniques. I have also examined different representations of geometry that make synthesis applicable, notably binary fields and distance fields. The actual work on geometry synthesis was limited and moderately successful, but generated interesting leads to investigate. In the time since, I have followed new developments in texture synthesis and would like to incorporate those in the study as well.

Author's Resume

An abridged version of the author's resume can be found below:

EDUCATION	**Rice University,** Houston, TX B.S. in Computer Science expected May 2004. GPA 3.86/4.00
EXPERIENCE	**National Instruments**, Austin, TX *Software Development Intern, Embedded Systems, May 2003 – August 2003* Modified the LabVIEW Embedded environment to generate multi-threaded C code. **Rice University**, Houston, TX *Software Developer, Programming Languages Technology, August 2002 – May 2003* Developed the programming environment DrC#. **Rice University**, Houston, TX *Research Assistant, Computer Graphics, May 2002 – December 2002* Independently researched and implemented texture and geometry synthesis. **Rice University**, Houston, TX *Teaching Assistant, Java and Design Patterns, January 2002 – present* Presented tutorials, consulted college students and graded their assignments.
PUBLICATIONS	Nguyen, D., Ricken, M., and Wong, S. "**Design Patterns for Marine Biology Simulation"**. To appear in *Proceedings of ACM SIGCSE 2004*.
HONORS	Sid Richardson Fellow, Tutor Tau Beta Pi Engineering Honor Society, Officer Louis J. Walsh Merit Scholarship in Engineering 2001 – 2004 Rice University President's Honor Roll Fall 2000, 2002; Spring 2002, 2003

CONCLUSION

The proposed work promises to reduce the time necessary to create meaningful 3-dimensional architecture for industrial design, electronic entertainment, and training by automating large portions of the creation process. Computer gaming and simulations used for educational purposes would particularly profit from computer-synthesized geometry.

Additionally, the study expands our knowledge of the link between geometry and textures and might allow us to apply useful techniques developed with textures in mind to geometry as well.

REFERENCES

[1] Rollings, A., and D. Morris. Game Architecture and Design. Scottsdale: Coriolis, 2000.

[2] Terragen – Photorealistic Scenery Rendering Software. Home page. 22 Jul 2003. Planetside Software. 7 Oct 2003 <http://www.planetside.co.uk/terragen/>

[3] WorldBuilder. Home page. 6 Oct 2003. Digital Element, Inc. 7 Oct 2003 <http://www.digi-element.com/wb/wb.htm>

[4] Deguy, S., and A. Benassi. A Flexible Noise Model for Designing Maps. VMV 2001 Proceedings.

[5] Deussen, O., P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. SIGGRAPH '98 Proceedings.

[6] Allouche, J.-P., M. Courbage, and G. Skordev. Notes on Cellular Automata. Institut fuer Dynamische Systeme, Universitaet Bremen, Report Nr. 458, November 2000.

[7] Think Labyrinth: Maze Algorithms. Home page. 21 Sep 2003. Walter D. Pullen. 7 Oct 2003 <http://www.astrolog.org/labyrnth/algrithm.htm>

[8] DungeonMaker. Home page. 2002. Dr. Peter Henningsen. 7 Oct 2003 <http://dungeonmaker.sourceforge.net/>

[9] Wei, L.-Y. Texture Synthesis by Fixed Neighborhood Searching. Stanford University, November 2001.

[10] Wei, L.-Y. Deterministic Analysis and Synthesis using Tree Structure Vector Quantization. Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing, 1998.

[11] Liang, L., C. Liu, Y-Q. Xu, B. Guo, and H.-Y. Shum. Real-Time Texture Synthesis by Patch-Based Sampling. ACM Transactions on Graphics, Vol. 20, No. 3, July 2001, Pages 127-150.

[12] Lorensen, W. E., and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. ACM Computer Graphics, Vol. 21, No. 4, July 1987, Pages 163-169.

[13] Ju, T., F. Losasso, S. Schaefer, and J. Warren. Dual Contouring of Hermite Data. SIGGRAPH 2003 Proceedings.