RICE UNIVERSITY SCHOLARS PROGRAM
COVER SHEET FOR PROPOSAL

PROJECT TITLE	Geometry Synthesis	
NAME	Mathias Ricken	DATE 10/07/03
RICE ID	0812203	MAILING ADDRESS
E-MAIL	mgricken@rice.edu	6360 Main St. Houston, TX, 77005
PHONE	(713) 348-1940	
ADVISOR	Dr. Joe Warren	
DEPT.	Computer Science	
PHONE	(713) 348-5728	TOTAL FUNDS REQUESTED:
E-MAIL	jwarren@rice.edu	$1675

ABSTRACT OF PROPOSED STUDY

The creation of meaningful geometry for architecture or simulations currently takes a considerable amount of time. Traditional geometry creation 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 it.

By automating large portions of this using the parametric generation of large-scale geometry and the subsequent addition of detail using techniques borrowed from texture synthesis, this process might 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 raw architecture.

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

TABLE OF CONTENTS

INTRODUCTION

The use of computer-generated, 3-dimensional geometry today is ubiquitous. It has been used for over a decade in *computer-aided design* (CAD) of commercial products or yet unbuilt 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.

The creation of meaningful geometry for these architectures or simulations, however, currently takes a considerable amount of time. Traditional geometry creation usually entails the making of a rough layout of the architecture, the manual addition of geometric detail, and a final application of colors and textures to it. 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 could reduce this figure significantly and open up new opportunities. By automating large portions of the traditional procedure described above, the designers would have more time to tailor the simulation to their specific needs, for example. In computer gaming and training, fast automatic generation of new environments would also increase 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: that of textures and that of geometry. It would prove that synthesis techniques developed with textures in mind 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.

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, are severely limited, or are not inherently related to geometry, but serve 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 change the terrain by applying filters that smoothen, roughen, or erode the ground, changing the sea level, or adding 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]. The distribution of the different plant species is determined by cellular automata, as found in the *game of life* [6],

Figure 1: Terragen Landscape

Figure 2: Plant Ecosystem

for example. Certain plant species 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 and 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).

It is obvious, though, that these methods of generating geometry function on a completely different scale than the ones I am proposing. These algorithms create entire continents and planets; they simulate the constructive forces of nature, not those of man. Even though human settlements could possibly be simulated using an ecosystem similar to the one used for plants, this would again happen on a planetary scale. It 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 is shaping a planet; it therefore cannot be simulated using the same means.

It should be possible, though, to combine the two approaches and 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. This, however, is a subject that will require further investigation.

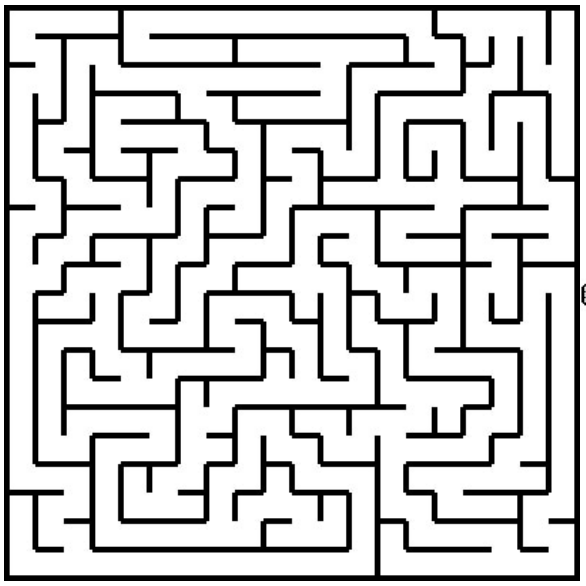

Figure 3: Mephistophelean Maze

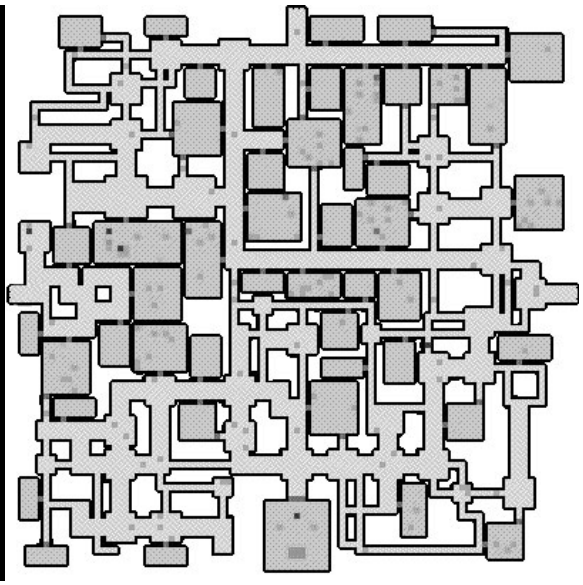

Figure 4: DungeonMaker Maze

Maze Generation

There are some algorithms for creating smaller structures [7]. One of them, recursive backtracking, involves randomly removing walls from a grid and moving into that direction if that cell has not been previously reached. Once a dead end is encountered, that is a cell where no wall can be torn down since all of the surrounding cells have already been stepped into, the algorithm 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 they do not reflect human architecture. They are too repetitive, restrictive, and use space in an awkward way. Additional shortcomings are that they can only be created on some kind of regular grid, and that they are mostly 2-dimensional in nature.

The DungeonMaker program employs a different stragety [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 an algorithm like the one described above look more realistic. They contain corridors, halls, and rooms that might serve some purpose, as shown in Figure 4.

Currently, the algorithm still operates on a 2-dimensional grid, sharing some of the disadvantages of Mephistophelean mazes. I am certain, though, that the procedure can be modified to make free use of space in 3-dimensions.

Texture Synthesis

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

Li-Yi Wei described an algorithm that is often used in texture synthesis in his doctoral thesis [9]. It requires a source texture 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 and looks at its *neighborhood*, the set of pixels that surrounds it. It then compares the neighborhood in the template texture to all possible neighborhoods in the source, and selects the one that is

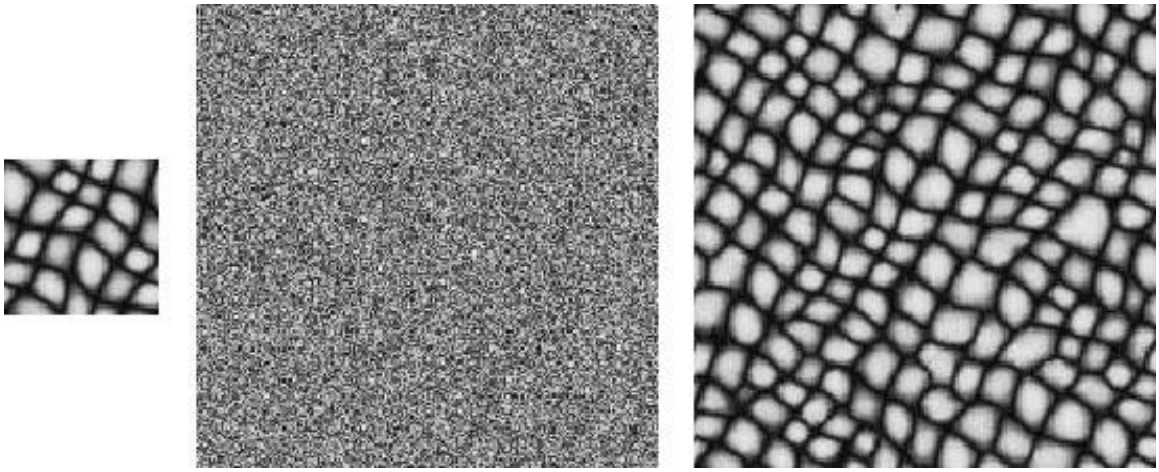

Figure 5: Texture Synthesis (left: source texture; middle: noise; right: synthesized texture)

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, large aberrations are punished more harshly than small ones, resulting in closer matches. Once the closest neighborhood in the source texture has been found, the pixel in the center of that neighborhood is copied into the current pixel in the template, and the algorithm continues with the next pixel.

Neighborhoods can vary in size and shape. Larger neighborhoods are able to reproduce larger features in the source texture. The neighborhood size should be proportional to the size of the features in the source, and 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 it should only take those areas of the texture into consideration that have already been synthesized. Since the areas above and to the left of the current pixel have already been processed, a causal neighborhood generally has the shape of an "L" rotated 90 degrees clockwise. The requirement for causality therefore determines the shape of the neighborhood.

The template, however, does not have to contain noise. It might start off with meaningful information or be composed of both that and noise. In this case, the algorithm will attempt a *constrained texture synthesis*. It is constrained since the synthesized pixels

must perfectly blend in with the ones already present, some of which can be to the right or below 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 it and make it larger.

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 it compares every neighborhood in the template with every possible neighborhood in the source, and every pixel in a neighborhood with its 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 proposed 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 scaled down, and synthesis begins at a much lower resolution. Since the size of the largest feature is also decreased, a smaller neighborhood is now able to capture it. The results of the low-resolution synthesis are then scaled up again, and the algorithm continues at the higher 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, a tree of neighborhoods is constructed before synthesis begins. Search time can therefore be 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-bases texture synthesis, the source is broken down into patches that are pasted into the template in their entirety. When a new patch is selected, only the borders are compared. This dramatically decreases 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 the geometry is in is divided into a uniform grid and the distance 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 are bad at creating large-scale structures. This was somewhat expected, though, since texture synthesis techniques require the image to be stationary and local, as explained in Wei's thesis; their application to large-scale geometry thus fails for the same reasons it fails in producing *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 further pursue this direction.

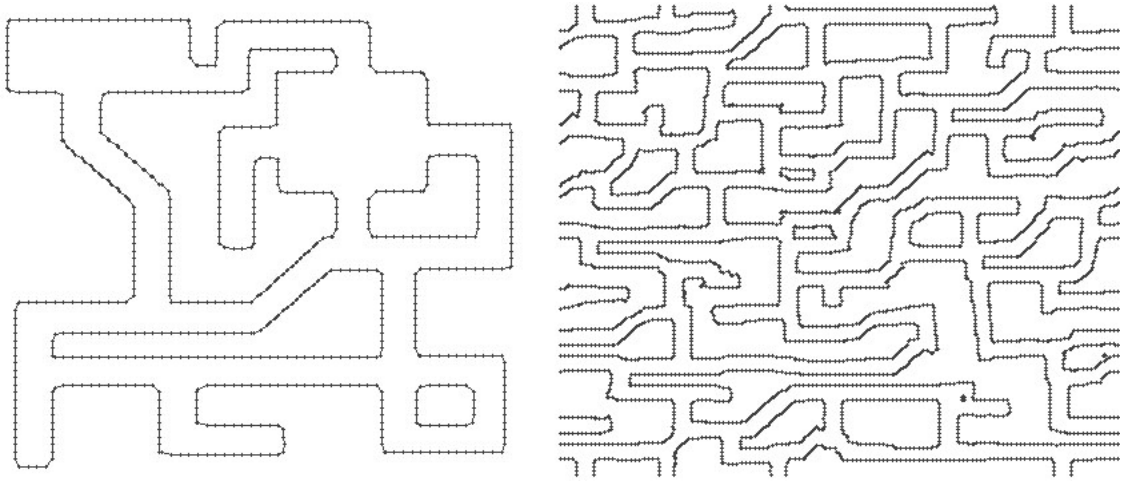

Figure 6: Geometry synthesis by analogy (left: source; right: synthesized)

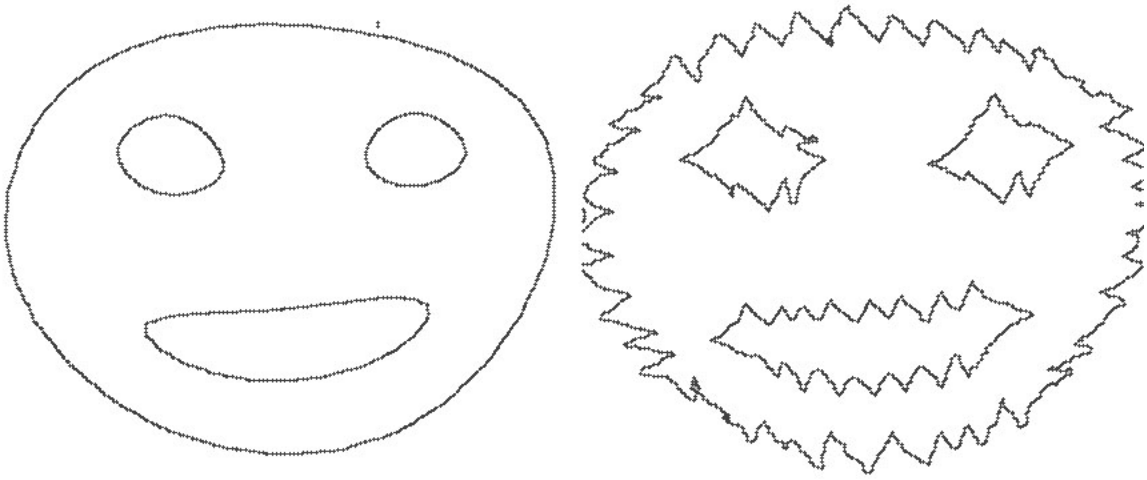

Figure 7: Surface properties modified by synthesis (left: source; right: synthesized)

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 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 it 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. I expect good results from this algorithm, even though I believe it will be hard to find the parameters that will lead to architecture as we know it today. The synthesized geometry should still be interesting, diverse, and suitable for use in a computer game, for example. 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.

Should the proposed algorithm unexpectedly not perform well in our modified environment, I will implement an alternative algorithm that has two stages itself. The first stage places points according to some of the parameters mentioned above. Depending on how close the points are, some of them will then be connected 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 algorithm described above has been proven to yield good results on a 2-dimensional grid and therefore seems more promising. I also I will be able to implement it more quickly than the untested algorithm, giving me more time to focus on geometry synthesis by analogy.

Once the raw geometry has been created, algorithms borrowed from texture synthesis will be used to 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 it. 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 it.

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 worked with in 2002 to work with 3-dimensional geometry. Transitioning from 2-dimensions to 3-dimensions will degrade runtimes even further and demand that I implement some of the optimizations that the works in texture synthesis describe. I expect particularly good results from using patch-based synthesis. Since I have not worked with these particular optimizations, I cannot make any statements about the time the editing system will require to synthesize a level; my intuition, however, tells me it to be in the range of several hours. The time to generate the raw geometry will be much shorter and the program will need no user interaction while it is synthesizing, so using this system will still be beneficial to designers.

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 it, my main interest is to prove that algorithms for texture synthesis can be used on 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 improving the synthesis result, though. 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. This 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. The algorithm could be modified to run several passes in areas where the energy remains above some threshold value.

Another possibility is to encourage the selection of pixels from adjacent neighborhoods. If the last pixel was generated from one source neighborhood, the neighborhoods surrounding it 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. This will make it possible to use the geometry in other programs and possibly make further modifications.

Budget

For the completion of this project, I request funds for the items below. I would also demonstrate my results at the 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 far 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.

Software		
	IntelliJ IDEA Editing Environment for Java, Academic Version	$100
Sources		
	Copying and printing; access fees to papers; books (to be donated to Fondren Library)	$150
SIGGRAPH		
	Conference registration	$425
	Travel	$700
	Lodging (4 nights)	$300
Total		**$1675**

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 2004

Submission of poster to SIGGRAPH 2004

August 2004

Presentation of work at SIGGRAPH 2004

FUTURE WORK

This project leaves many directions open for further investigation. If the parametric creation of large-scale geometry is to generate architecture as we see it every day, the algorithms and parameters used here will certainly need additional research. A cooperation with researchers from the fields of architecture and sociology might prove to be fruitful here.

Another option is to improve the algorithms used in the synthesis by analogy part. As mentioned before, optimizations are even more critical in 3-dimensional space than they are in the two dimensions texture synthesis operates in. It should therefore be useful to study all the optimizations that have been found to work for texture synthesis and to apply them to geometry.

It is also possible that geometry makes it possible to employ special optimizations unsuitable for textures. The search for those would certainly be worthwhile. One area that could provide insight is that dealing with compression of geometric data. To reduce the amount of information, 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, it 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 geometry synthesis might reduce the number of comparisons necessary to find the best-matching neighborhood and thus result in improved runtimes.

AUTHOR'S QUALIFICATION

During the summer of 2002, I conducted initial research on geometry synthesis under the supervision of Dr. Warren. In the advent 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 source code for different operating systems and hardware platforms. **Rice University**, Houston, TX *Software Developer, Programming Languages Technology, August 2002 – May 2003* Developed the programming environment DrC#. Advisor: Dr. Cartwright **Rice University**, Houston, TX *Research Assistant, Computer Graphics, May 2002 – December 2002* Independently researched and implemented texture and geometry synthesis algorithms in computer graphics; developed applications for a haptic input device. Advisor: Dr. Warren **Rice University**, Houston, TX *Teaching Assistant, Java and Design Patterns, January 2002 – present* Presented tutorials on Unix, Java, design patterns, and tools; consulted college students and graded their homework assignments. Advisor: Dr. Nguyen
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, Andrew, and Dave 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, Sébastian and Albert Benassi. A Flexible Noise Model for Designing Maps. VMV 2001 Proceedings.

[5] Deussen, Oliver, et al. 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, Li-Yi. Texture Synthesis by Fixed Neighborhood Searching. Stanford University, November 2001.

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

[11] Liang, Lin, et al. Real-Time Texture Synthesis by Patch-Based Sampling. ACM Transactions on Graphics, Vol. 20, No. 3, July 2001, Pages 127-150.

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

[13] Warren, Joe, et al. Dual Contouring of Hermite Data. SIGGRAPH 2003 Proceedings.