Accelerating the Scalable City

Sheldon Brown, Kristen Kho, Kwangyoon Lee, Erik Hill, William Huber

The Scalable City: aesthetic and conceptual goals.

The Scalable City[1] is a series of artworks by Sheldon Brown and his Experimental Game Lab. Members of this lab have included Alex Dragelescu, Mike Caloud, Joey Hammer, Erik Hill, Carl Burton, Kristen Kho, Kwangyoon Lee, Daniel Tracy and William Huber. The artworks of The Scalable City consist of prints on paper and canvas, computer animated videos, procedural animations, multi-channel video installations, and interactive 3D computer graphic environments. The initial content of The Scalable City is built via a data visualization pipeline. In this pipeline, data is taken from existing cities – satellite imagery, GIS data, statistics, photogrammetry – and is subjected to simple algorithmic transformations, creating a new city whose forms are applicable to our algorithmic future. Each step in this pipeline builds upon the processes which precede it, amplifying and exaggerating features, artifacts and patterns.

The Scalable City consists of 5 major components - landscape, roads, lots, architecture and vehicles[2]. The process of developing each of these begins with data captured from networked real-world applications. This raw data, drawn from a real world referent, is transformed by an algorithm that creates a characteristic imprint of its own. In the case of the landscape, the captured data comes from satellite imaging. This visual representation of the landscape is transformed by a simple process of duplication, rotation, copying and pasting; the process creates a new landscape which retains naturalism in its details, but with a high level of algorithmic decoratism in its large scale structure.

Road systems are then “grown” into this landscape. First, an analysis determines suitable areas for occupation. These areas are demarcated by an encircling roadway and then space-filling curves are grown using an L-system graph. L-systems are recursive, self similar forms and are used for such things as the simulation of plant structure. The curves are derived from Archimedes spirals. The roads only intersect at their branching points, never crossing, ending in cul-de-sacs. The pattern created is a decorative spiral

labyrinth, evocative of art nouveau iron grates, illuminated manuscripts, and other decorative uses of space-filling patterns. These curves are turned into roads with appropriate texture maps, including the demarcation of intersections and sidewalks. Each time the road system is generated, the specific end-result is unknown.

The goal is a work which reflects and illuminates the character of the built environment and the transformation of lived space in the contemporary period. The processes which transform the landscapes of The Scalable City are what Alex Galloway has described as allegorithms: algorithmic procedures which are offered to the player for interpretation[5]. One of the more important goals of The Scalable City is a networked, interactive system that affords the player/viewer a chance to participate in and experience the generation of these transformed landscapes using the conventions and literacy's reminiscent of gamic spaces. In the pursuit of this goal cell processing has been investigated as a solution to the increasing computational demands of the project.

The Scalable City articulates the aesthetic concept of *troiage*, as coined by Brown[3]. This aesthetic has emerged from Brown's history of creating interactive 3D computer graphic artworks. It posits the combination and transformation of pre-existing codes, conventions and aesthetics from painting, cinema and architecture. The combinatorial and abstractive powers of the computation afford, and require, the development of such new aesthetic principles, and troiage grounds them in both the materiality of the production of code and the specific visual rhetorics drawn from those practices which have been remediated digitally.

A digital image, or any representational data-structure drawn from some relational process with the non-digital world, can be reinterpreted endlessly. A digital picture of the Mona Lisa can be turned into a bit stream for playing on a sound synthesizer, or it can form a behavior system for a flock of micro robots or it can be the control code for the Tokyo traffic light system: once information has been captured from the world, it is mutable and available for indefinite re-working. Brown's troiage aesthetic foregrounds this tension between the desire to simulate an increasingly remote referent and an opposite desire for transmutation and the exercise of algorithmic power.

Software development in the context of art practice

Software development in the context of the practice of contemporary art has distinctive features which distinguish it from commercial or scientific practice[4]. Rather than being driven by client requirements or experimental design, code written for a project such as The Scalable City must meet the broad aesthetic, conceptual and rhetorical goals of its authors. Software-based art practitioners can respond to the discovery of new affordances in the hardware and software tools with which they work. Singular and idiosyncratic solutions can be deployed without concerns about manufacturing or experimental reproducibility.

This flexibility is not unbounded, however, and often it is those very same conceptual and aesthetic objectives which compel the authors to resolve obstinate and difficult problems, with particular attention to issues of the coherence of the participant's experience and the exhaustive hermeneutics that might be engaged against the work. Artists often are compelled to be innovative in order to achieve singular effects. In comparison, innovation in traditional game development is often hampered, particularly after the initial design stage, by its organizational constraints and market requirements. Because the aims of The Scalable City are first and foremost conceptual, and the material products of the work (prints, animations, interactive installations) are variform, the artist enjoys the freedom to integrate novel technologies, or to use existing technologies in unexpected ways. As a result, project such as The Scalable City present an opportunity to explore unexpected affordances in new technologies, and simultaneously to develop tools and libraries to overcome limitations that may not have been predicted by the designers of those technologies. This general approach to software development as a means of art practice, frames the approach to integrating cell processing into the The Scalable City project.

Toward procedurality and interactivity

The early iterations of The Scalable City involved the modeling and rendering of all assets in a time-intensive process. The goal, however, is a real-time interactive work which generates its assets on demand.

Future plans for The Scalable City feature multiplayer creation of cities drawn from a range of real-world captured data. This movement, from the rendering of assets which will be stored and integrated into the work to the procedural generation of assets in real-time with the interactive experience, speaks to a current problem in videogame development.

The rich graphical power of contemporary display hardware, and the mushrooming capacity of distribution methods(from floppies and cartridges to CD-ROMS to DVDs to Blu-Ray disks and broadband networks) have created an inflationary demand for content. This has increased game development costs, even as margins decline and competition increases. In 1982, a game like Activision's Pitfall could be designed, developed and programmed by one individual in less that 1000 person-hours and sell well over 4 million copies, recent games involve production teams of dozens of modelers, world designers, musicians, programmers and other personnel. The ballooning costs of game development have made publishers risk-averse, and innovation is stifled both by the scale of production and by the sense of risk.

The procedural generation of assets does not only reduce production time by removing modeling and pre-rendering stages from the production pipeline, but it also reduces costs and makes the distribution of visually rich content over the internet viable. It also binds together more tightly the authorship of interactive behaviors and the visual style of the work, improving aesthetic coherence. Finally, the emergent character of procedurally generated content can respond more richly to the computational environment, customizing the player-participant's experience. Although the goals of The Scalable City are not those of even an independent game designer, much less a commercial game developer, the conceptual nature of the work provides another incentive for a trajectory toward procedurality: it is a reflection of the very allegorical ambitions of the work, a reflection on the emergent and procedural transformation of space.

Initial Cell encounters

In working with emergent technological approaches, we will always encounter unexpected obstacles, and one of the values of our activities is helping to ferret out conditions in new technological arenas that inhibit their use in diverse scenarios. Initial conditions such as reorganizing system directories and files and

compilation problems with our preferred C++ environment took some time to address, as problems with the sdtio.h header and libspe.h header were tracked down and resolved by writing a wrapper around the headers.

During this initial period of working with the CP system, we found that because of its very unique architecture, we needed to develop a stronger understanding of its practical advantages and shortcomings. This discovery period should lead us to better program design, as we proceed, and it is influencing directions that we think are the most viable in our overall goal of accelerating our gaming applications. The initial goals, in particular, were to develop working models and methods for managing SPU control threads, using DMA control blocks, managing DMA operations on the SPUs, and partitioning data structures to the SPUs. Our methodology was experimental and empirical rather than theoretical: we designed a series of tests to determine optimal loads given the computational overhead created by delegating processes to the SPU, as well as drafting "facade" classes to encapsulate the complexities of DMA access in CP.

After determining the apparent computational costs of simple operations incurred by spawning SPU threads, as well as dot product and vector operations performed by the SPU, we wanted to reliably manage the frequency of SPU thread initializations. We found it helpful to develop our own libraries for encapsulating CP operations. Called the Cell Architecture Framework and Extensions (CAFE), the libraries were geared toward applications utilizing heavy loads of 3D vector math and mesh data manipulation.

The road problem

Having developed an at least provisional model of the affordances offered by Cell Processing, we then set out to determine an actual process within the Scalable City to implement using the new system. Of the various components of The Scalable City, we looked for a problem that needed to be solved to allow the project to achieve its conceptual goals as a real-time, generative and interactive project, yet was still computationally constrained and well-bounded.

The initial iteration of The Scalable City requires much of its environment data, particularly the roads and landscapes, to be pre-processed and made available to the drawing application. The roads are central to the

conceptual and rhetorical aims of the work, reflecting the emergent transformation of real landscapes by algorithms of human desire and distributed agency. They are also, and inseparably, key to the visual aesthetic effectiveness of The Scalable City: space-filling curves have a storied role in art and design history.

Road generation in The Scalable City is an L-system of Archimedes spirals. A spiral is generated and then placed so that its origin lies on an existing road. (Figure 1) The initial road is a perimeter road around the entire cityscape and is constructed via a series of computer vision techniques in combination with the landscape height-map data. The generation of the perimeter road is a constant-time procedure and computationally. However, the placement of increasing numbers of spirals within the constraints determined by the artist is not.

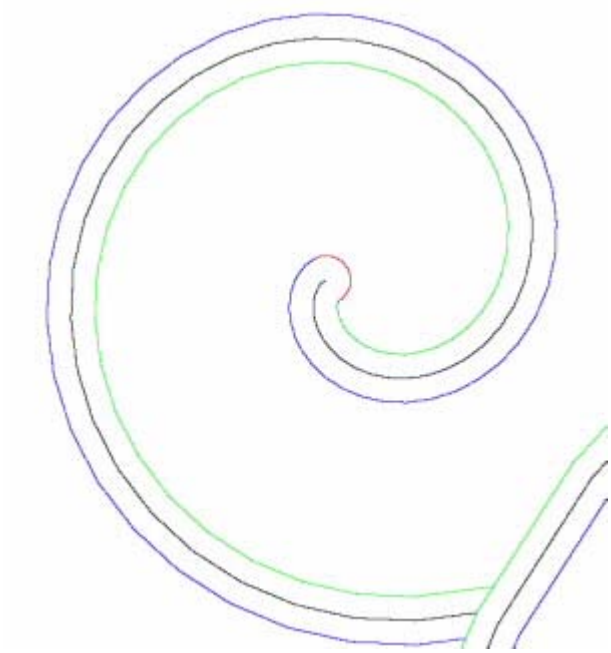

Figure 1 A spawn curve consists of a center curve (black), left offset curve (blue), a right offset curve (green) and a cap curve (red).

In the procedure by which road curves are added to a landscape, the algorithm, in either its original, single-processor mode or its subsequent implementations, 1. chooses a starting point on an existing road and 2. generates an Archimedes spiral curve, (Figure 2.) which we refer to as a challenger curve. We 3. check if the challenger curve intersects any of the already existing curves. 4. If (a.) the challenger curve does intersect an

existing curve, we go back to step 1 and try a new challenger curve. If (b.) the challenger curve does not intersect an existing curve, we insert the curve into our list of valid road curves and go back to step 1, for a defined constant number of iterations

Figure 2 Early result where roads were tested for intersections with each other but no rules were set. A circle is used in place of actual border curve data.

This algorithm is run repeatedly for a number of possible challenger curves, and the result is a pattern similar to the one in figure 1. This is a simplification to the actual algorithm used; in practice, the aesthetic requirements of The Scalable City dictate that we set rules in place to control the frequency of placement, initial curve-spawning angles, and other parameters (Figure 3.) Those additional rules add a constant overhead to the algorithm.

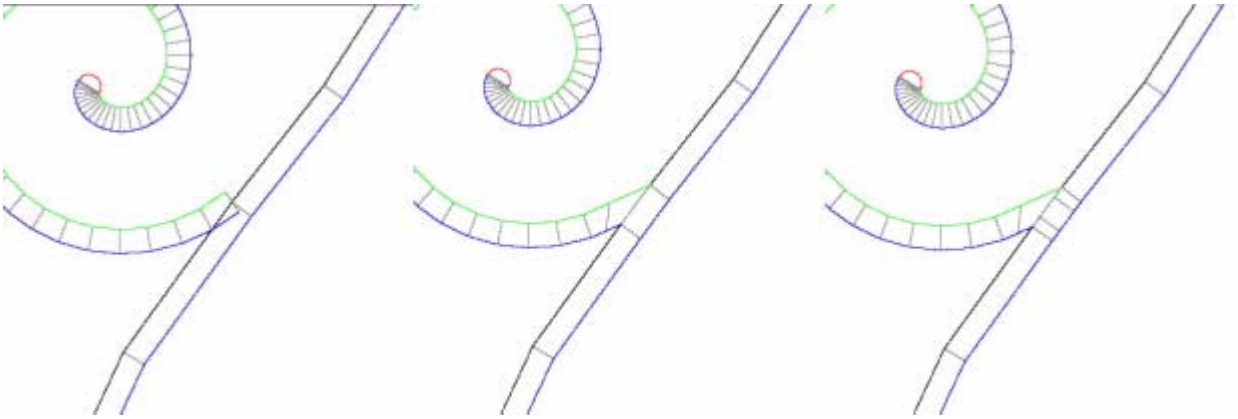

Figure 3 Process of conforming curves to intersections. Spawned curve on the left attaches to road curve with new vertices added to each, including points for sidewalks.

The core element to the algorithm outlined above is its testing for intersections with prior road segments. The brute-force nature of the curve-curve intersection test over the monotonically increasing number of road curves is a computationally expensive operation, and the most expensive component is a data-parallel problem.

During the intersection testing, none of the results are mutually dependent, which makes it an ideal candidate for our foray into cell processing. In our implementation, we decided to split the data by curves as a convenience for an intuitive partitioning, in keeping with the knowledge-transfer constraints of the project. Thus, we divide the number of current road curves amongst the SPUs (making sure we properly handle uneven cases when the number is not a multiple of the number of SPUs requested) and then publish the data to DMA. We also must use DMA to make the challenger curve available to all SPUs. Then we run our curve-curve intersection testing and return the number of intersections found to the PPU. The PPU has the task of summing up the results and making the final decision as to whether the challenger curve is a valid road curve, or if it intersects an existing road curve and must be discarded.

The PPU is allocated specific tasks within a loop for a defined constant maximum tries: it 1. generates a challenger curve (a candidate for insertion into the map) as a series of line segments with an origin on an extent road segment. The DMA address of that challenger curve is then 2. sent to each of the SPUs. It then 3. partitions the current list of inserted road-curves to the SPUs, and 4. receives the count of intersections

between the challenger curve and the extant road-graph. It 4. sums the number of intersections. If there are no intersections after all the SPUs have returned their final count, then 5. it inserts the curve into the list of curves (the road-chart); otherwise 6. it tags the curve as invalid and does not insert it.

The control variable is the defined MaxTries constant, which is the only mechanism that ends the loop. This is an aesthetic decision: it introduces the possibility of a very sparse map, although the probability of a dense map increases logarithmically with the value of the constant. (See Figure 4) It would of course be possible to change this procedure by forcing the loop to continue until a minimum or maximum number of successful insertions had occurred: the decision not to do so is tied with the rhetoric of the work about the nature of algorithmic desire and emergent, yet unpredictable behaviors.

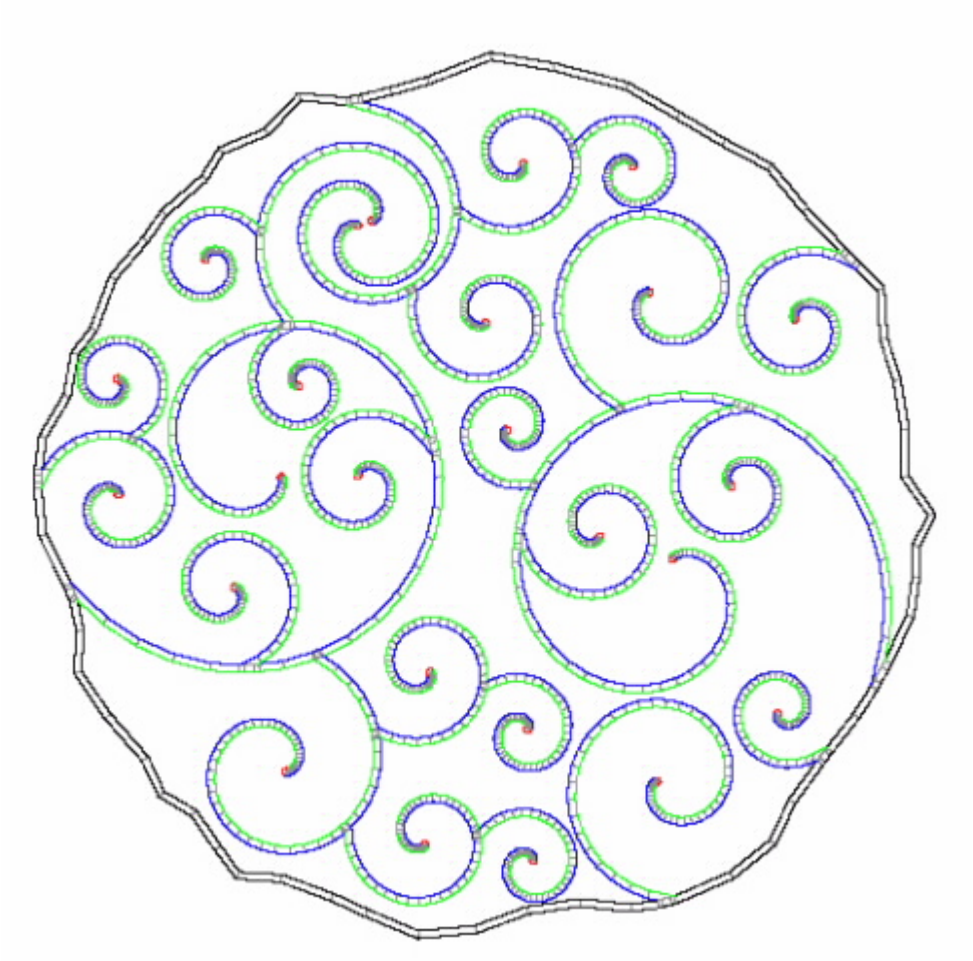

Figure 4 Example of completed road system of moderate density.

Each SPU is allocated the following series of tasks: they each 1.receive the challenger curve from the PPU, along with 2. a subset of the list of curves which constitute the existing road systems. For each of the curves in that subset, the SPU 3. tests for an intersection with the challenger curve; if it finds one, it will 4. increment an IntersectionCount variable (which has an initial value of 0.) When it has tested against all curves in the sublist, it 5. sends the value of IntersectionCount to the PPU

We created simple TCP client and server programs to transfer commands and data to and from the Cell. The Python library interface for sockets makes this relatively quick and simple. The client program sends the desired parameters (random seed, number of tries, road width, etc.) and border curve data to the server. The server uses it to generate the roads and then sends the results back to the client. While our Python client suffices for now, the Scalable City application may eventually incorporate its own client code.

Performance

Generating a road structure comparable to the original Scalable City output is relatively fast – between 0.5 and 2.5 seconds. The original implementation, using the MAYA API and MEL (Maya Embedded Language) scripts, took a number of hours to complete the road structure. Since more than half of the total time is spent on intersection testing, it decreases almost linearly with the number of SPUs used. Each blade has two Cell Processors, and each Cell Processor has 8 SPUs. In addition, there are certain algorithm level and low-level optimizations that have been explored, such as quad-tree organization of spatial data, which proved to not provide any speed increase. (Table 1)

These timings include the time spent writing the results to file. Results may vary slightly depending on the parameters specified. Note that the number of tries is not synonymous with the number of SPU intersection tests because our algorithm will abandon a road as soon as it is found unfit. Also, increasing the number of tries does not guarantee that more roads will be generated. Since the program will attempt to insert the largest curve size first, more large roads will be inserted and this will decrease the area that the more numerous smaller curves can fit into.

Floating Point(s)

One issue emerged with the large scale used in Scalable City coupled with the limitations of using single precision floating point on the Cell. This caused some of our intersection tests to fail. (Figure 5) To work around this problem, without having to convert all of our operations to double precision, we decided to normalize all of our values based on the border curve data we received from the Scalable City application. After our computations are complete, we de-normalized the values to retrieve the original scale, and then wrote the results to file.

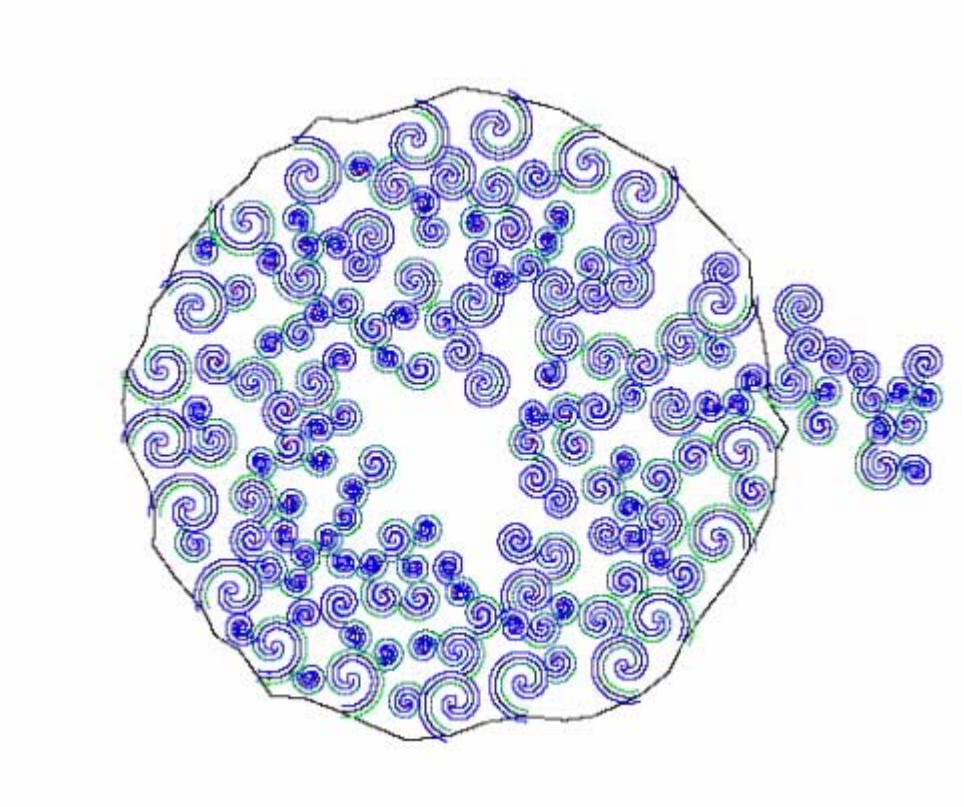

Figure 5 Single precision floating point error allows curves to leak out of intersection tests near the x-axis.

Conclusion

With these results, we are now integrating Cell computations into our application, and have resolved the most significant computational bottleneck in the application. With this solution, we are able to make our terrain and database extensible in the interactive application, as opposed to pre-computing the data. Increasing the efficiency of these operations is a pre-requisite to the next iteration of the project: a networked, highly multi-user, real-time application.

These results meet the highest expectation that we had for this project, and validate our initial hypothesis. The Cell has proven to be a powerful computational solution for data parallel applications. While it does present particular programming challenges, it has proven itself invaluable in helping The Scalable City achieve its conceptual goals.

Figure 6 Road system incorporated into full Scalable City application.

References

[1] Brown S and Experimental Game Lab. The Scalable City. http://crca.ucsd.edu/sheldon/scalable [5 September 2004]

[2] Brown S 2004. Scalable city 0.7a. In *Proceedings of the 12th Annual ACM international Conference on Multimedia* (New York, NY, USA, October 10 - 16, 2004). MULTIMEDIA '04. ACM, New York, NY, 989-990. DOI:10.1145/1027527.1027764

[3] Brown S 2004. “Troiage Aesthetics.” In American Association of Artificial Intelligence Proceedings, Washington DC

[4] Brown S 2004.“Ersatz- A Software Platform for Making Art”, International Society on Virtual Systems and MultiMedia Japan, 10/04.

[5] Galloway A 2006 *Gaming : essays on algorithmic culture.* University of Minnesota Press, Minneapolis

Tables

	1000 tries	10000 tries	100,000 tries
8 SPUs	2.008266 seconds	2.371372 seconds	3.481144 seconds
16 SPUs	1.224655 seconds	1.451451 seconds	2.226997 seconds
# roads generated	23	25	24

Table 1. Total Execution Time for Road Generation on the Cell