From the Ground Up: Building a Machine City for *Matrix: Revolutions*

Charles Rose*
Tippett Studio

1 Introduction

The development and production of The Matrix Revolutions' machine city sequences were challenging on most every front. From the beginning, the machine city had been described as a huge living fractal / coral reef, twice the size of the greater LA basin, alive in every aspect: building itself on the fly, swarming with inhabitants, alive with lighting, electrostatic events, and atmospherics. Even the massive towers were alive in their slow purposeful movements as they tracked events within the city. How were we to design, layout, build, and render such a city, without a map?

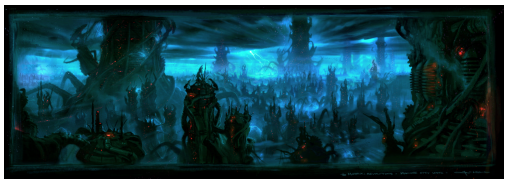

Figure 1: Machine City key art

2 Approach

The scale and complexity of the city led us to use a combination of techniques: a procedural growth system for a base, combined with hand dressed building and towers used for larger details and areas close to camera. The first step was to build shared libraries of component pieces, building blocks that could be combined either procedurally, or by hand to grow potentially thousands of variations of buildings. These libraries were categorized by building type, so that in general one would build from components within the appropriate library, which included ground, tower, magma pit, power cables, and city block libraries. Each library was built to four different resolutions: high, medium and low resolution nurbs models were intended for rendering at the appropriate LOD, and extremely low res polygonal models were constructed for layout and visualization inside of maya. All library components had to be built conforming to specific rules to ensure that they would be properly animated and swapped out at render time for the proper resolution. The procedural aspects of the layout evolved considerably over time. Early versions used texture maps to control what type of buildings belonged in a particular neighborhood, as well as parameters such as density, height, width and the number and size of tower branches. Fx Lead Dan Rolinek's final implementation of the procedural city grew builder curves along paths. These curves carried all pertinent information about the buildings to be grown, and particles dropped from the curves to the surface would birth individual building components. Force fields were then added to control which direction to bend the towers and how to orient ground buildings to create the organically grown look we desired. Hand dressed buildings were built using the very same component libraries, using similar rules. These conventions allowed animation applied on the appropriate nodes to be inherited by RIB archives. Since most camera moves were relatively small compared to the scale of this massive city, we were able to hardwire archive resolutions into the scenes with simple grouping and naming conventions, rather than worrying about tuning procedural LOD settings.

*e-mail: rose@tippett.com

3 Rendering

All library components were processed into individual renderman RIB archives which would be shared by all shots and processes. Since these archives had to incorporate multiple reiterations of modeling and shading changes, a carefully constructed workflow was essential to ensure that this part of the pipeline would keep up with the many changes look development required. All RIB archives were called from the master RIB by using DelayedReadArchives, whose ability to load archives into memory only when visible was the only possible way to render this kind of heavy dense material. Our initial problems with DelayedReadArchive were centered around being to able to edit specific shading parameters on the fly at render time. This was important as there were many animated reflection maps in surfaces, many of which were shot and frame specific, e.g. lightning flashes in the sky and city would be projected back into buildings as animated reflection maps. Our solution was to edit "user" attributes in the master RIB at render time to pass messages to shaders in the RIB archives that had been written to accept the appropriate attribute messages. Arbitrary output variables were used extensively in the rendering; usually some 18 variables were output in addition to the main render. Typical stream outputs would include not only such standard shading outputs as diffuse and specular, but specific LightSource lighting group components, several different reflection passes, z depth, y depth, per building masks based on a user attribute, and masks based on textures for glows and bioluminescence.

Figure 2: Final Composite

4 Conclusion

Carefully laying the groundwork of well organized libraries in the beginning of the process allowed us to determine on a shot by shot basis how much to rely on procedurally placed components versus hand dressed architecture, without worrying about shifts in any aspect of the look. The reliance on either approach could be varied shot to shot, depending on the larger context, without disrupting the overall look. Creating a rich stream of output variables from the shaders allowed us to modify many of the shading components in the final composite.