Procedural Generation of Mediterranean Environments

N. Mikuličić* and Ž. Mihajlović*
* University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
niko.mikulicic@gmail.com, zeljka.mihajlovic@fer.hr

Abstract - This paper describes an overall process of procedural generation of natural environments through terrain generation, texturing and scattering of terrain cover. Although described process can be used to create various types of environments, focus of this paper has been put on *Mediterranean* which is somewhat specific and has not yet received any attention in scientific papers. We present a novel technique for procedural texturing and scattering of terrain cover based on cascading input parameters. Input parameters can be used to scatter vegetation simply by slope and height of the terrain, but they can also be easily extended and combined to use more advanced parameters such as wind maps, moisture maps, per plant distribution maps etc. Additionally, we present a method for using a satellite image as an input parameter. Comparing results with real-life images shows that our approach can create plausible, visually appealing landscapes.

Keywords: procedural modeling, landscape generation, virtual environments, natural environments

I. INTRODUCTION

Reproducing realistic environments has always been a challenge in computer graphics mainly due to a large number of data needed to be simultaneously processed and displayed on screen. With many optimization methods and ever advancing graphics hardware, realistic landscapes have become achievable in real time and are widely used in computer games, military simulators and film industry. Recent trends show that virtual landscapes have also found their usage in touristic promotions and since environmental research has become an important topic in the last years, it can be expected for virtual environments to have a major role in simulating how changes in different environmental parameters influence the surrounding environment.

Manually creating a whole environment would be a long lasting, tedious task, therefore many procedural techniques have been developed to aid the process. Although procedural, these methods usually have many parameters and finding the right values is often a time-consuming trial and error process. In recent years, various methods for inverse procedural modeling have been introduced [7][22]. Rather than setting parameter values manually, these methods provide ways to learn them. Nevertheless, procedural modeling remains the most common approach to generation of environments.

Authors discussing generation of natural virtual environments usually reproduce mountain meadows and dense forests. Although visually appealing, tall and dense trees efficiently hide everything behind them so many objects can be safely culled and popping effects can be easily hidden. In this paper we focus on natural Mediterranean environments which have not yet been discussed. Mediterranean environments often lack tall trees and instead have very dense shrubbery. Since shrubs often cannot hide the landscape behind, various optimization techniques have to be implemented with extra care for popping effect to be properly hidden and for the environment to be efficiently rendered in real time.

Our generation process is being done in four steps: preparation, terrain mesh generation, texture mapping and scattering of terrain cover. In preparatory step, a user defines a set of textures and models to be used and assigns them input parameter values. The second step is used to create a terrain triangle mesh from heightmap which is followed by procedural generation of splat map and texturing of the terrain in the third step. The last step places the terrain cover using scattering algorithms to achieve natural randomness. Although divided into four steps, the user often returns to preparatory step to adjust parameter values and then repeats the process until he is satisfied with results. The whole process needs to be repeated only when user changes the terrain geometry. Otherwise, user can focus on just one step.

Although the focus of this paper is on Mediterranean, the described process can also be used to reproduce any other natural environment simply by using different textures and models.

After presenting related work in chapter two, chapter three continues by explaining techniques used to represent terrain, textures and terrain cover as well as the optimizations required for real time performance. Fourth chapter focuses on algorithms used for procedural texturing and scattering of terrain cover. We present our results in chapter five and give conclusion and some guidelines for future work in chapter six .

II. RELATED WORK

This work belongs to procedural modeling of natural environments, and as such, it connects various fields of computer graphics.

Procedural modeling automates the process of content creation by using set of rules and algorithms rather than creating content manually. It is often applied where manual creation of content would be too cumbersome a task. Procedural modeling has been successfully used to create various types of content some of which include textures [6], plants [10], buildings [12], cities [18], and terrains [8].

Terrain generation is often being done in two steps: generation of heightmap and creation of 3D mesh. Heightmaps are usually generated using fractal algorithms [1][11], noises [1][6] and erosions [1][13][17]. After creating a heightmap, terrain mesh is generated and optimized using level of detail algorithms. Static LOD algorithms reduce computational at the cost of memory demands and often need nontrivial and rigid preprocessing stage [23]. Continuous LODs are generated dynamically and are therefore more flexible and scalable but also computationally more demanding [15][21].

Forest representation includes representations of individual trees as well as the process of achieving their natural distribution in ecosystem. Trees have been represented with level of detail [3], billboards [20], parallel billboards [9], volumetric textures [16] and billboard clouds [5]. To create an ecosystem, trees are scattered using *global-to-local* approaches [4][14] that define some global rule by which the scattering occurs. Another approach is *local-to-global* [2][4][14] that models interaction between individual plants from which the global distribution arises.

III. RENDERING

Natural environment consists of several somewhat independent layers: terrain, textures and terrain cover. In order to achieve efficient rendering, optimization should be done in each layer.

Terrain is usually represented with a heightmap which can be generated procedurally or downloaded from Internet in case real-world data is needed. Fig. 1 shows heightmap of Croatian island Ist (left) and heightmap generated using Perlin noise and Fractional Brownian motion (right). To generate terrain triangle mesh from heightmap we use Lindstrom-Koller simplification [15] on a single terrain region. Creating multiple regions with levels of detail is left for future work.

Textures are applied to terrain by simple planar texture mapping. Splat mapping technique is used to provide more surface details close to observer while simple color mapping is applied to more distant areas where details are not visible anyway.

Terrain cover is split into two subcategories: details and high cover.

Details are small objects, like grass, visible only from short distances. Due to their size, many objects are needed to cover any portion of a terrain so they tend to use up most of the available computational resources. In order to achieve real-time rendering of large fields covered with grass, various optimization methods have to be implemented. Modeling each blade of grass independently would quickly reach a maximum of vertices and polygons

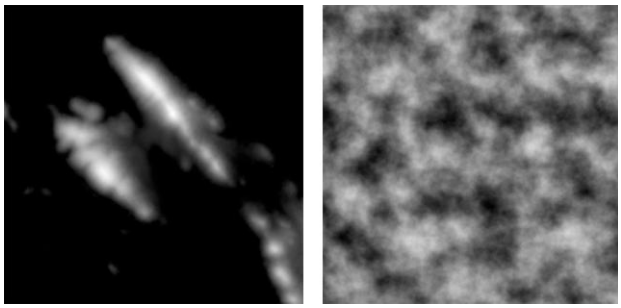

Figure 1. Heightmap of Mediterranean island Ist, Croatia (left) and procedurally generated heightmap (right)

a graphics card can process in real time and for that reason, more efficient approaches have been developed. We represent multiple blades of grass with one axial billboard. Many billboards are needed to cover an area with grass and sending each billboard to GPU independently would mean too many draw calls and therefore, slower rendering. Instead, multiple billboards are grouped together and sent to GPU as point clouds in which every vertex represents the position of a single billboard. Billboard geometries are then created at runtime in geometry shader at positions defined by vertices in the given point cloud. Additionally, since details are visible only from short distance, all billboards with distance to observer greater than some user defined value are culled. Terrain split to regions would fasten the culling process, allowing large groups of billboards to be discarded with a single distance check. To avoid sudden popping effects, a transitional area is used in which a billboard goes from completely invisible to fully visible using alpha cutout technique.

High cover includes all objects that can be viewed from greater distances like trees or larger rocks. To optimize the number of polygons in a scene, levels of detail have been used. Objects close to the observer are rendered using high quality models, while lower quality models are used on objects that are further away. To avoid popping effects while transitioning between different models of the same object, a cross-fade technique is used. Both models are rendered for some small amount of time, while one is slowly fading in and the other is fading out. For the lowest quality model of a tree we use a simple axial billboard.

IV. PROCEDURAL GENERATION

In this chapter we focus on procedurally transforming input parameters into texture weights and terrain object placement probabilities. Texture weights are used to create splat maps and color maps for terrain texturing while terrain object placement probabilities are used to generate terrain cover using scattering algorithms.

A. Weights Calculation

Input parameters are defined in preparatory step. For each input parameter, every texture and terrain object defines minimum and maximum values and a weight function to describe parametric area it resides in. For example, a texture can be given a slope input parameter. To define a parametric area it resides in, a texture has to specify minimum and maximum slope. Weight function is used to describe texture's weight or preference in area

between minimum and maximum. Fig. 2 represents a common weight function although any curve can be used. Weight functions are defined in $[0, 1]$ interval and scaled to fit specified range.

Once we have defined parametric areas for all textures and terrain objects, their weights can be easily calculated. For texture or terrain object t with parameters p_i and weight functions w_{p_i}, the weight w_t is defined as:

$$w_t(\boldsymbol{r}) = \prod_{i=1}^{n} w_{p_i}(f_{p_i}(\boldsymbol{r})), \quad (1)$$

where n is the number of parameters and f_{p_i} is parameter dependent function which maps terrain position $\boldsymbol{r}$ to parameter value. It can represent calculating surface slope, elevation, sampling value from texture etc. Put simply, final weight is calculated by multiplying weights of individual parameter values at given position.

B. Texturing

Texture weights are calculated to give us information how much each texture belongs to specified area. This information can be used to procedurally texture a terrain. For terrain texturing we used splat mapping and color mapping techniques.

Color mapping technique textures a whole terrain with just one texture. One pixel of color map usually covers not as small portion of terrain so this technique usually results with dull and blurry terrains when observed from close. It would require extremely large textures to create detailed surface using this technique which would be quite inefficient.

To add more surface detail, splat mapping technique is used. Splat mapping uses multiple high detailed surface textures that are mapped to a small part of terrain and then tiled to fit the rest. One additional texture called splat map is mapped to whole terrain and used to provide information on how much each surface texture contributes to the final color of surface. Final color is determined dynamically in fragment shader by sampling all surface textures and multiplying sampled colors with their contributions sampled from splat map.

Splat map can be directly generated from weights defined in previous chapter. One pixel in splat map corresponds to terrain area at position $\boldsymbol{r}$. Texture weights are then calculated using (1), scaled to sum of 1 and stored in a single splat map pixel, one channel per surface texture. This means that one splat map can store weights for four surface textures. We can always use a second splat map for four new surface textures, but that can become inefficient. Usually, four surface textures are more than enough.

Color map is generated similarly. Instead of storing texture weights and calculating surface color dynamically in fragment shader, colors are sampled, mixed and baked into a color map. To avoid high frequencies in resulting texture, colors are not sampled from original surface textures but rather from their last mipmap.

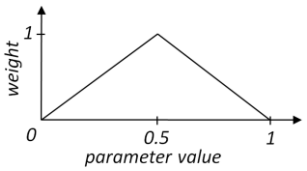

Figure 2. Weight function

C. Scattering of Terrain Cover

After procedurally texturing a terrain, next step is to provide it with grass and high cover.

To achieve natural randomness and uniform distribution of grass, we use a regular grid with random displacement algorithm. Terrain is split into a regular grid with one grass billboard assigned to the center of each cell. Each billboard is then displaced in random direction by amount small enough to remain inside the given cell. For each billboard position $\boldsymbol{r}$, grass weights are calculated using (1). Grass type is then chosen using roulette wheel method where weight of grass type is proportional to associated probability.

Potential collisions of grass on cell borders can be ignored as they visually do not make much difference. Collisions between trees, however, should be avoided. Trees can have different sizes and placing them in a regular grid would not be as easy as it was with grass. Therefore, for scattering trees we use simple random scattering with collision detection. We randomly choose a position, calculate tree weights and choose a tree sort using roulette wheel method. If chosen sort cannot fit onto position without colliding with surroundings, we remove the sort from consideration and try with another. The algorithm ends when all trees have been placed or the maximum number of iterations has been reached.

In reality, forest is not just trees being randomly scattered. They are grown from seeds, fighting for space and resources. Stronger trees prevail giving birth to new ones and the process repeats. New seeds fall in the vicinity of mother tree which results in trees than are not just randomly scattered but rather grouped. To simulate this process we have implemented a survival algorithm with environmental feedback similar to algorithm described in [2]. At the beginning of each iteration, all plants generate seed at random positions inside seeding radius. Seed that falls outside of terrain or on a ground that is not inside specie's parametric area is removed. Next step is detecting collisions between plants. Plants in collision are fighting for survival and weaker plant, a plant with smaller viability [2], is eliminated. At the end of iteration, all plants increase in age and plants that have reached their maximum age are removed from the ecosystem. The algorithm ends after defined number of iterations.

Fig. 4 shows comparison of these three scattering algorithms where different tree species are being represented by circles of different colors and radii. Grid displacement algorithm (left) gives good uniform distribution of objects of the same size. Random scattering with collision detection (middle) can successfully scatter objects of different sizes, however, it does not produce clusters of same species like survival algorithm (right) which gives forest a more natural look.

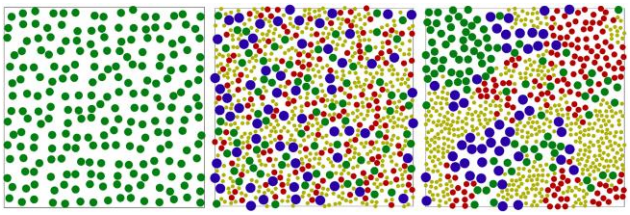

Figure 4. Scattering algorithms. Scattering on regular grid with random displacement (left), random scattering with collision detection (middle) and survival algorithm (right)

D. *Classifying Satellite Image*

For procedural texturing and scattering of terrain cover, we can also use a satellite image as an input parameter. To translate image into weights we use a simple, color based classification. To calculate weight at certain terrain position, satellite image is sampled and given color mapped to HSV space where boundaries between colors are more obvious.

Fig. 3 shows a satellite image with its classification. Due to atmospheric scattering, images taken from great distances have their colors shifted towards blue. This makes classifying harder and more advanced terrain cover classifier would be needed to fully indentify the coastal line or tell difference between sea and vegetation. More advanced methods for classifying surface cover have been developed [19], however this approach is simple and gives good enough results in natural, uninhabited areas.

After color sampled from satellite image is classified to surface cover type, we can treat that information as weight equal to 1 if terrain object is of specified type, or 0 if it isn't.

V. RESULTS

To evaluate our approach we will make comparisons in different scales between real and procedurally generated Mediterranean landscape.

Fig. 5 shows Croatian island Ist textured using different input parameters. Fig. 5 (left) represents terrain textured using slope and height. This method is generic and usually gives good results on both real and procedural terrains. Fig. 5 (middle) represents the same landscape textured using terrain cover classification from satellite image. Terrain is textured more precisely using this method, however it can only be used on real-world

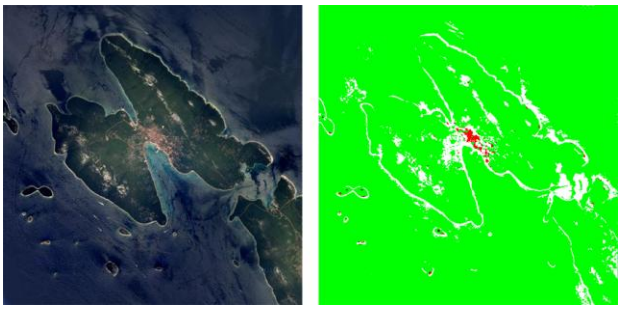

Figure 3. Satellite image of Ist (left) and classification of terrain cover (right)

landscapes. Additionally, procedurally generated terrain is never a perfect representation of real one due to inaccuracies in heightmap, interpolations and simplifications of terrain geometry. For that reason, coastal line is often misaligned as it can be seen on Fig. 5 (middle). Fig. 5 (right) tries to solve the coastal line problem using combination of previous two approaches. The coastal line height is defined and everything above that level is textured using terrain cover classification while everything below using only slope and height as input parameters. This makes coastal line more monotonous but consistent.

Fig. 6 shows satellite image of smaller, uninhabited part of Ist and its procedural representation generated using already mentioned coastal line reconstruction approach. The technique proves to be more reliable in uninhabited areas with image taken from closer distance. In addition to textures, Fig. 6 (right) also contains trees scattered according to terrain cover classification.

Following image (Fig. 7) shows the same terrain from the ground. To represent Mediterranean grass we used textures that are similar to following Mediterranean plants: *Lactuca serriola*, *Trisetum flavescens*, *Conyza sumatrensis*, *Eupatorium cannabinum*, *Urtica dioica* and *Melilotus sp*. Also, a model of *Arbutus unedo* has been used as a bush. To have more control over distribution of each species we included additional spread parameter for every grass and tree type.

Finally, Fig. 8 displays a comparison between real and procedurally generated landscape. Every object on the image is procedurally scattered except for the two bushes in front which are placed manually for easier comparison.

Figure 5. Comparison of procedural texturing by slope and height (left), by classifying satellite image (middle) and using coastal line reconstruction approach (right)

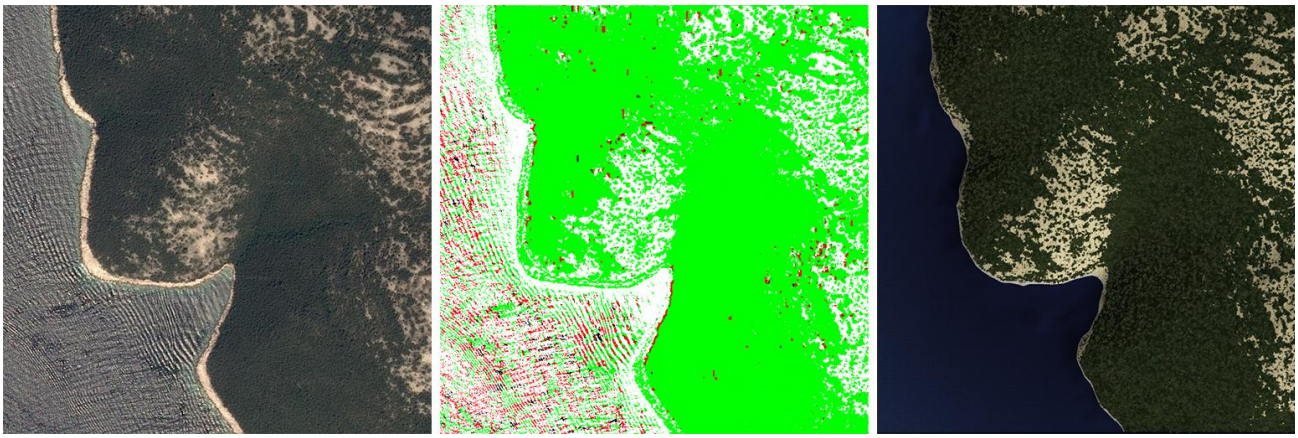

Figure 6. Satellite image of Dumboka cove at Ist (left), terrain cover classification (middle) and procedurally generated landscape (right)

Figure 7. Procedurally generated Dumboka cove from ground

Figure 8. Comparison between real (left) and procedurally generated landscape (right)

VI. Conclusion and Future Work

We presented a method for procedural generation of environments through terrain generation, texturing and scattering of terrain cover using cascaded input parameters. By calculating and multiplying weights of individual input parameters we obtain information about how much each texture or species wants to reside on given terrain position. That information can be used as texture's contribution to terrain color when texturing the terrain, or as probability when scattering the terrain cover. Height and slope of the terrain have proven to be reliable input parameters for generic uses. In case of real-world terrains, simple terrain cover classification from satellite image can be used to provide basic information for texturing and scattering of terrain cover. To fix coastal line issues caused by geometry misalignments between real and generated terrain, we introduced a coastal line reconstruction approach. Additionally, we used a spread parameter to have more control over distribution of each species. Comparing real life images with those generated procedurally, we believe that our method can create plausible and visually appealing Mediterranean landscapes.

Many suggestions for future work can be made. Terrain should be split into regions for faster view frustum culling of terrain geometry, grass and trees. Level of detail algorithms should be used to dynamically reduce the polygon count of faraway terrain regions. Those regions could also use cheaper texturing technique, for example color mapping instead of splat mapping. Closer, high detail regions could use normal mapping or even fractal displacement of geometry to achieve 3D look of Mediterranean karst. Scattering terrain cover based on satellite image would benefit from more advanced terrain cover classifier. Although height and slope as input parameters give good generic results, they are not primary factors in shaping Mediterranean environment and it would be interesting to see how would additional parameters, like wind map and resistance to wind and salt, affect the final look of the environment.

References

[1] T. Archer, "Procedurally generating terrain", 44th annual midwest instruction and computing symposium, Duluth, 2011., pp. 378–393.

[2] B. Beneš, "A stable modeling of large plant ecosystems", In Proceedings of the International Conference on Computer Vision and Graphics, Association for Image Processing, 2002, pp. 94–101.

[3] C. Colditz, L. Coconu, O. Deussen, C. Hege, "Real-time rendering of complex photorealistic landscapes using hybrid level-of-detail approaches", 6th Interational Conference for Information Technologies in Landscape Architecture, 2005.

[4] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz, "Realistic modeling and rendering of plant ecosystems", In Computer Graphics (Proceedings of ACM SIGGRAPH) (1998), pp. 275–286.

[5] X. Décoret, F. Durand, F. X. Sillion, J. Dorsey, "Billboard clouds for extreme model simplification", In ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH) (2003), pp. 689–696.

[6] D. S. Ebert, S. Worley, F. K. Musgrave, D. Peachey, and K. Perlin, Texturing & Modeling, a Procedural Approach, Elsevier, 3rd edition, 2003.

[7] A. Emilien, U. Vimont, M.-P. Cani, P. Poulin, and B. Beneš, "WorldBrush: Interactive example-based synthesis of procedural virtual worlds", ACM Transactions on Graphics (SIGGRAPH 2015), vol. 34, issue 4, pp. 11.

[8] J.-D. Génevaux, É. Galin, É. Guérin, A. Peytavie, and B. Beneš, "Terrain generation using procedural models based on hydrology", ACM Transactions on Graphics (SIGGRAPH 2013), vol. 32, issue 4, 143:1–13.

[9] A. Jakulin, "Interactive vegetation rendering with slicing and blending", In Eurographics 2000 (Short Presentations), 2000.

[10] R. Měch, and P. Prusinkiewicz, "Visual models of plants interacting with their environment", In Computer Graphics (Proceedings of ACM SIGGRAPH) (1996), pp. 397–410.

[11] G. S. P. Miller, "The definition and rendering of terrain maps", In Computer Graphics (Proceedings of ACM SIGGRAPH) (1986), vol. 20, pp. 39–48.

[12] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural modeling of buildings", In ACM Transactions on Graphics (SIGGRAPH, 2006) , vol. 25, issue 3, pp. 614–623.

[13] F. K. Musgrave, C. E. Kolb, and R. S. Mace, "The synthesis and rendering of eroded fractal terrains", In Computer Graphics (Proceedings of ACM SIGGRAPH) (1989), pp. 41–50.

[14] B. Lane and P. Prusinkiewicz, "Generating spatial distribution for multilevel models of plant communities", In Proceedings of Graphics Interface ’02., vol. 1, pp. 69–80.

[15] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner, "Real-time, continuous level of detail rendering of height fields", In Computer Graphics (Proceedings of ACM SIGGRAPH) (1996), pp. 109–118.

[16] F. Neyret, "Modeling, animating and rendering complex scenes using volumetric textures", IEEE Transactions on Visualization and Computer Graphics, vol. 4, issue 1 (1998), pp. 55–70.

[17] J. Olsen, "Realtime procedural terrain generation", Technical Report, University of Southern Denmark, 2004.

[18] Y. I. H. Parish, and P. Müller, "Procedural modeling of cities", In Computer Graphics (Proceedings of ACM SIGGRAPH) (2001), pp. 301–308.

[19] S. Premoze, W. B. Thompson, and P. Shirley, "Geospecific rendering of alpine terrain", In Rendering Techniques (1999), pp. 107–118.

[20] J. Rohlf, J. Helman, "IRIS performer: A high performance multiprocessing toolkit for real-time 3D graphics", In Computer Graphics (Proceedings of ACM SIGGRAPH) (1994), pp. 381–394.

[21] F. Strugar, "Continuous distance-dependent level of fetail for rendering heightmaps (CDLOD)", In Journal of Graphics, GPU, and Game Tools, 2009, vol. 14, issue 4, pp. 57–74.

[22] O. Šťava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, and B. Beneš, "Inverse procedural modelling of trees", Computer Graphics Forum, 2014, vol. 33, issue 6, pp. 118–131.

[23] T. Ulrich, "Rendering massive terrains using chunked level of detail control", SIGGRAPH Super-Size It! Scaling Up to Massive Virtual Worlds Course Notes, 2002.