WEB-BASED & INTERACTIVE 3D PICTORIAL MAPS WITH DATA-DRIVEN PROCEDURAL PLACEMENT

A PREPRINT

Hernaldo Henriquez
Departments of Visualization
Texas A&M University
College Station, Texas 77831
hjhenriq@gmail.com

Ergun Akleman*
Departments of Visualization &
Computer Science and Engineering
Texas A&M University
College Station, Texas 77831
ergun.akleman@gmail.com

August 13, 2021

ABSTRACT

People have been telling stories for thousands of years. They help to communicate events in a way that is easy to understand because we can relate to them. But to fully understand a story, one must comprehend the context. Like in what time, place and culture things happen. Pictorial maps help connect the place with the events in a visual way. We have created a prototype tool that would help artists create 3D interactive pictorial maps using data-driven Procedural Placement. It is data-driven meaning that it takes data as input and generates new data as output without the need of changing the code. Input data are image maps (density maps, height maps, road maps, etc) and JSON data (ecotope definitions), all of them are manually created by the artist and they are intuitive and easy to control without the need of learning complex software or programming languages. We follow the idea of ecotopes used in the development of the video game Horizon Zero Dawn, where ecotopes define a group of assets and how frequent each one appears. We also created an algorithm that uses road maps to orient buildings to face the nearest road in an efficient manner. In our case we use this system to represent ancient cities from the Bible to explore how that could help people to close the gap between the present context and the context in which the events happened.

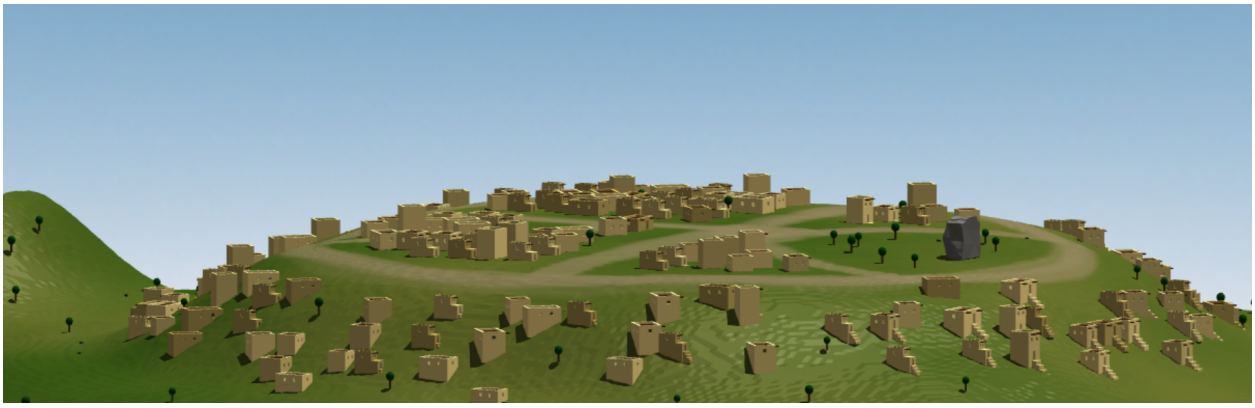


Figure 1: *A view of a procedurally created city from our engine*

*website:people.tamu.edu/ ergun

1 Introduction and Motivation

Stories have been told for thousands of years. They have been used for different purposes, like entertainment, or to explain certain events or concepts. Research has shown that stories can communicate a message more effectively than using just facts with no narrative. That is because in stories we are involved emotionally and we can be "transported" into them [6].

To fully understand a story and be immersed in it, one has to understand the context of the story. And there are different types of context, for example the historical, a story happens in a certain time, a period in history, or the cultural context, the story happens in a specific culture with specific beliefs, religions, foods, traditions, etc. There is also a physical context, the story may happen in a specific place. A story that happens in New York City may be very different from one that happens in Cuzco, Peru. The environment is different.

Pictorial Maps are devices that help connect what happens with where it happens. They are maps that have illustrations that display relevant information of places in the map, for example landmarks, flora, fauna, landscape, events, etc. They don't need to be accurate, just representative, so they are often made in a cartoon style with simple forms. Pictorial Maps can help storytelling in conveying the context of a place with only one image.

Figure 2: *Example of a pictorial map of Alaska from Ruth Taylor White (1930's)*

For this project we decided to explore the development of a tool that would make the creation of Pictorial Maps easier. Specifically, we wanted to use the idea of Pictorial Maps to help explaining stories from the Bible. Usually Bibles come with maps at the back of the book, but the readers hardly understand the context of the story, because they are not familiar with it (readers are in a very different cultural, historical and physical context than the one of the events that happened) and those maps don't help conveying that context.

Figure 3: *Example of a map from a Bible*

We wanted to develop a prototype tool that would help artists create simple depictions of ancient cities. It had to be intuitive, easy to use and give them control.

2 Previous Work

Most of the pictorial maps we can find are paintings or drawings. But there are also examples of them in three dimensions. Naz made 3D interactive pictorial maps of different countries. In our project we follow this idea, making 3D web-based interactive maps of ancient cities. We also use simple models for the buildings, following a caricature aesthetic for effective depiction [9] [12].

The use of virtual environments to help tell stories has been explored before. For example, in 1996, Disney created an attraction in one of their parks where people could explore the world of the movie "Aladdin" in Virtual Reality [11]. In our case we think that similarly a virtual representation of ancient cities will help our users to immerse themselves in the stories of the Bible.

We decided to use a procedural approach for the creation of our ancient cities, using procedural placement of predefined assets.

There has been previous work in the creation of virtual worlds using procedural methods and there are surveys that the reader can look at to find more about them [14] [4]. Some applications have been the creation of terrain, vegetation, rivers, roads and cities.

For procedural creation of cities, there has been previous work using L-systems to create building models and to place them. One project reconstructed the old city of Pompeii using design rules and information from the ground plans [8] and another project created urban cities from image maps as input (elevation maps, water maps, population density maps, etc.) [10]. There has also been work to create Ancient Roman and Hellenistic buildings using Procedural Modeling [13]. Another group has worked in GPU-Based real-time distribution of vegetation in a terrain using height maps, slope maps, water maps, moisture maps, and more [2]. Galin et. al. have proposed a method for the procedural creation of roads in a given scene with a given start and end point for a road. Their method takes into account different parameters such as slope, obstacles like rivers and trees and shortest distance [5]. And another project went further creating roads and settlements of villages in arbitrary terrains [3].

For the creation of the open-world Video Game Horizon Zero Dawn, the team at Guerrilla Games created a system for Procedural Placement of assets that would help speedup the process of map creation, placing objects like trees, bushes, rocks, etc. in real time using a brush, giving the artist control [16]. This works by introducing two main concepts: **density maps** and **ecotopes**.

Ecotopes are groups of related objects that define an ecosystem. For example, a forest ecotope might have trees, fern, bushes, etc. Each of them with their own density (how often they are present in the forest) and footprint (the size in space they use as a radius from their center).

Density maps are grey scale image maps where pixel value represents density, or in other words the probability that in the place represented by the pixel the ecotope will be present.

Each ecotope will have its own density map. Then to start placing assets, there is a discretization phase in which they run dithering over the density maps creating **placement maps** that are binary. For placement maps each pixel means whether the represented place has an asset from the ecotope or not.

This allows the creation of maps with assets placed randomly but guided by the artist through the use of density maps and ecotopes. This is something we decided to use as well, to rapidly create pictorial representations of cities. The fact that Pictorial Maps don't need to be exact replicas of real cities means that we could use randomness more freely. In our case we apply Procedural methods for the creation of cities but it is different from the rest in that other Procedural approaches generate cities modeling reality (like population density to decide where to put more buildings) but our approach is artist driven. The artist decides which kind of buildings appear more often and where through the use of density maps. Our program also lets the users create maps easily and intuitively because the users are not required to learn complex software or programming languages, they just need to draw gray-scale maps and make simple ecotope definitions in JSON format.

3 Methodology

We have developed a prototype tool that uses Procedural Placement to generate 3D Pictorial Maps from input data that is manually created by the artist. Our program uses that to place predefined assets in the map and gives them a specific orientation. That information is stored in a custom file system that uses the JSON format and is then given as input to a Web App that displays the resulting 3D Pictorial Map.

1. **Placement** We have a script that reads the input data which are density maps, height maps, road maps and ecotopes definition in JSON. Ecotopes have a hierarchy, on which the most important are placed first and then the rest won't be placed in the same space, so for example if there is an ecotope that defines houses above one that defines a forest, then there shouldn't be any trees placed on top of a house. To do that the program combines the density maps with mathematical operations. Density maps are read in order of priority and the current map is multiplied by the inverse of the previous one to combine them. We also use road maps that define the roads of the cities and are combined with density maps to avoid collision.

(a) *Example of a density map (it's a gray scale image)*

(b) *Road Map is combined with density map*

(c) *Example of a placement map (it's a binary image)*

Figure 4: *Density maps are converted to placement maps in the discretization phase*

For each density map representing an ecotope we run a discretize phase where we apply Floyd-Steinberg dithering, getting a binary placement map. This is used later in a placement phase, which iterates on the pixels of the maps. If a pixel is white the program will place an asset from the ecotope in the space defined by the pixel. The position inside the world is calculated by using a predefined pixel size that is in world units and the height map to define the altitude. The specific asset that will be placed is chosen randomly using the probabilities that the ecotope defines. We have also implemented options to allow scale, offset and rotation of assets that must be declared in the ecotope definition. Finally, the placement information is stored as JSON data in a *placement.json* file, which is an array where each object contains the asset ID, position, rotation and scale in world units.


```
{
  "name": "forest",
  "priority": 1,
  "footprint": 5,
  "data": [
    {
      "assetId": 2,
      "probability": 0.8,
      "footprint": 2,
      "allowOffset": 0.5,
      "allowScale": 0.3
    },
    {
      "assetId": 3,
      "probability": 0.2,
      "footprint": 0.5,
      "allowRotation": 6.28318,
      "allowOffset": 2,
      "allowScale": 0.3
    }
  ]
},
```


(a) *An ecotope definition in JSON*


```
{
  "assetId": 3,
  "position": {
    "x": -133.333,
    "y": 4.267,
    "z": -148.35
  },
  "rotation": 6.247,
  "scale": {
    "x": 1.007,
    "y": 1.007,
    "z": 1.007
  }
},
```


(b) *A placement definition in JSON*

Figure 5: *JSON data used in procedural placement*

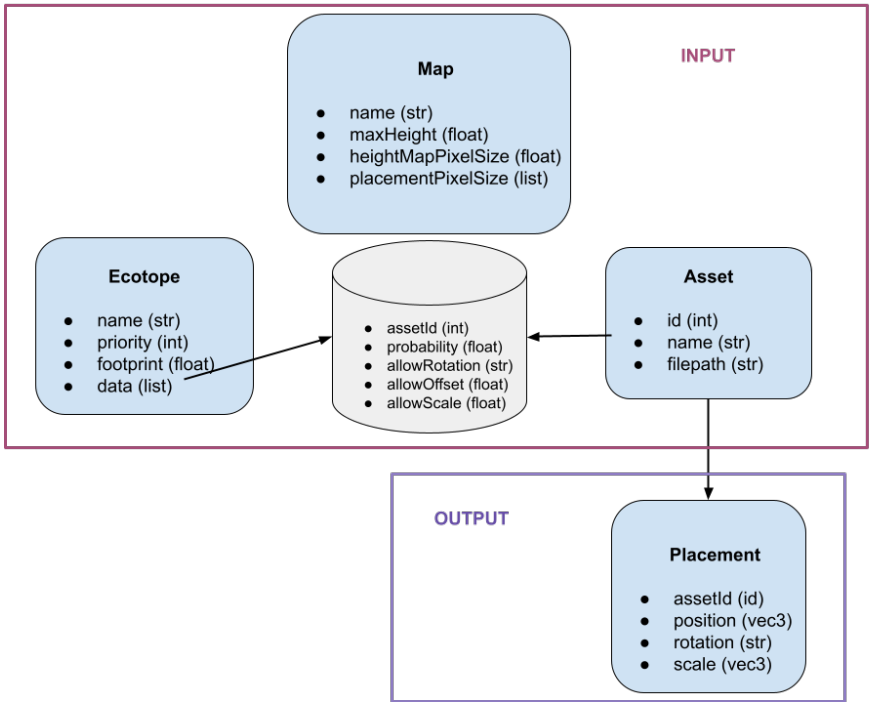
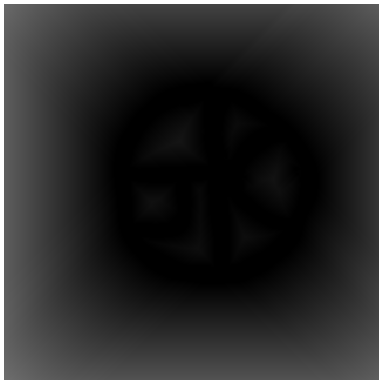


Figure 6: *Data used in the system. Input: Map configuration, ecotopes definition and assets list. Output: Placement data defined by asset ID and transform*

2. **Orientation** We use road maps to define the orientation of houses. We orient them so that the doors are facing the nearest road. We do this because orienting the buildings either completely randomly or all with the same orientation makes the city look not real. Making houses face to the nearest road is a challenging problem that can be translated to finding the nearest neighbor. A naive solution would have an order of magnitude of O(n^2) if n is the number of neighbors. For example we could take the pixel that represents the place where a particular house is located in the road map, iterate over all the pixels in the maps calculating their distance to the house and then choose the one with the minimum (only taking into account pixels that are in a road). Having to calculate that for every house would be quite expensive. We came up with a solution that is faster but we cannot guarantee it will always get the nearest road. But since pictorial maps don't need extreme precision, an approximation is good enough.

   We dilate the road map, adding the new pixels to an open list and the road pixels to a closed list. Then we iterate dilating the pixels in the open list, moving them into a closed list and adding the new dilated pixels to the open list. On each dilation, the new pixels are colored differently; if the road map starts with a color value of 0 for the pixels that represent being road, then we can apply 254 dilation operations colored differently (using 8 bit gray-scale) and if we increase the color value by one on each iteration, that means the pixels further from the road would have higher color values. The resulting map, we call it a **distance map**, where the value of the pixel means how far it is from the nearest road (in pixels distance). This process is similar to "flooding" a group of pixels, advancing one pixel distance at a time. Then applying horizontal and vertical derivative filters in that map would give us two maps that point to the direction to go, to get closer to the road. We call the union of those last two maps the **orientation map** where the red channel represents the horizontal derivative and the green channel represents the vertical derivative. With that, we have the necessary information to calculate the orientation of the houses and the complexity of our algorithm is linear in the number of pixels instead of quadratic, because every pixel is dilated only once (when they are in the open list). We have the limitation that if a pixel representing the location of a house is further than 254 pixels to the nearest road our algorithm won't calculate the correct orientation. That case is not probable to happen because our entire maps are 320x320, and houses are placed near roads.

(a) *Road map used*



(b) *A distance map (also known as Dijkstra map)*

(c) *Orient map, each pixel points to the nearest road*

Figure 7: *Maps used for orientation*

4 Implementation

The implementation was done with a Python script that reads the input data, manually created by the artist, then runs Procedural Placement and stores that information into a JSON file called *placement.json*. Then to visualize the generated map, we have developed a Web Application in JavaScript with the library three.js. That way, the 3D Pictorial Maps can be displayed on any device with a Web Browser. For asset creation we use Procedural Modeling in SideFX Houdini. We have created four different house models that are fairly simple, with a low triangle count and simple materials.

Our application is data-driven, meaning that changes in the input data will make changes in the output data, without the need of changing the code. The data that the Web App receives is surface.json, that has the information necessary to create the terrain of the map, surface.png which is the texture to be mapped to the terrain, placement.json contains the information to place assets in the map (asset ID, position, rotation, scale), and finally assets.json maps each asset's ID with their file location. The Web App works by first reading surface.json and adding two triangles for each pixel of the height map. Then the program reads placement.json and places assets one by one in the scene with the information given.

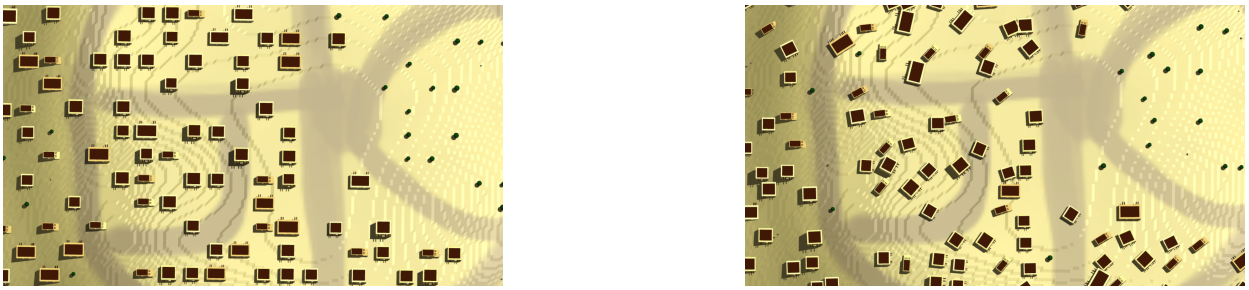
Our code is open source and publicly available in GitHub in the following link: `https://github.com/HenrYxZ/pictorial-map`

5 Limitations

One of the main limitations of this approach, are the patterns generated in the discretization phase. The map looks fake when the user can notice a grid pattern in the placement of assets. In real life, things aren't ordered perfectly so it's necessary to add some randomness to make things look more believable. To do this, we add random offsets in the placement, arbitrarily choosing a maximum range. But this also brings another problem to the table, it can happen that after applying the offset, the asset will be colliding with another one; and it is necessary to try with different values to find the right one. Our implementation doesn't deal with this problem yet.

Another problem we face is that assets have different sizes, so in the discretization phase, depending on the pixel size one may use, some assets might have too much empty space around or might not fit in the space represented by a pixel, potentially colliding with other assets around. For this problem we define a footprint for each ecotope which is a pixel size large enough to fit the biggest asset in the group leaving some margin. The problem is that the user has to define that footprint value, so it may be necessary to try with different values until one is good and this also limits ecotopes to have assets with similar footprints. Also, if the ecotope footprint is at least half the size of the map density pixel size, placement pixels will be divided by the integer result of that ratio, so that smaller assets can make use of empty space. A downside of that approach is generation of grid patterns.

Finally another issue we encounter is placing assets where the ground has slopes. The application places assets at the height of the respective pixel in the height map, and each pixel represents 1mt2 in the world. Our problem is that sometimes the terrain cuts off assets. To fix this one option could be to modify the triangles around the recently placed asset so that all of them are at the same height as the asset. Also houses can be oriented to have the door facing the lowest height.



(a) *Grid pattern generated in placement*

(b) *Placement using rotation and offset (0.5mt)*

Figure 8: *Comparison between placement with and without random offset and rotation*

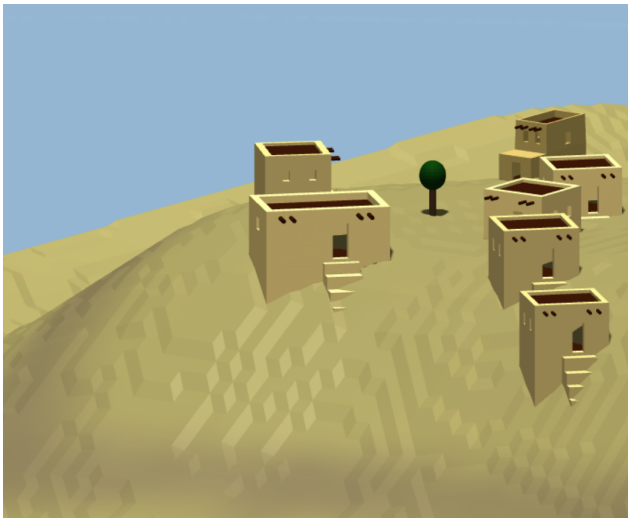


Figure 9: *House gets cut off by the terrain*

6 Future Work

There are many ways we think this work can be further improved. For example:

- An interface to change maps in real time. For that, Procedural Placement would have to be computed faster. The team at Guerrilla used Compute Shaders to achieve that.
- Create another feature to add walls, maybe using a walls map where the artist can easily draw walls around the city. Ancient cities were usually fortified so that feature would clearly help better represent them. The problem would be how to translate the pixels that represent walls to a world object. One option could be a voxels approach, another one could be having predefined objects for pixel windows of a certain size. Another problem would be what to do with slopes.
- An advantage of using a virtual pictorial map is that it can be displayed with different lighting conditions to communicate different moods. In the future, there could be lights added to the houses for example, to show the cities at night and add a skylight that would mimic different hours of the day. The Web Engine babylon.js has a sky material that can achieve that.
- In the future, these mini-maps could also be exported as a glTF or USD scene. That would allow them to be rendered using other applications like Unreal Engine or Blender, or they could be viewed using Augmented Reality.
- Finally another possible improvement could be to use more stylized shaders to render the scene as if it was a painting or a drawing, giving even more power to the artist to communicate the mood they want [15] [7] [1].

References

[1] C. Chan, E. Akleman, and J. Chen. Two methods for creating chinese painting. In *10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings.*, pages 403–412. IEEE, 2002.

[2] B. T. do Nascimento, F. P. Franzin, and C. T. Pozzer. Gpu-based real-time procedural distribution of vegetation on large-scale virtual terrains. In *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 157–15709. IEEE, 2018.

[3] A. Emilien, A. Bernhardt, A. Peytavie, M.-P. Cani, and E. Galin. Procedural generation of villages on arbitrary terrains. *The Visual Computer*, 28(6):809–818, 2012.

[4] J. Freiknecht and W. Effelsberg. A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction*, 1(4):27, 2017.

[5] E. Galin, A. Peytavie, N. Maréchal, and E. Guérin. Procedural generation of roads. In *Computer Graphics Forum*, volume 29, pages 429–438. Wiley Online Library, 2010.

[6] M. C. Green and T. C. Brock. The role of transportation in the persuasiveness of public narratives. *Journal of personality and social psychology*, 79(5):701, 2000.

[7] M. Justice and E. Akleman. A process to create dynamic landscape paintings using barycentric shading with control paintings. In *ACM SIGGRAPH 2018 Posters*, pages 1–2. 2018.

[8] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. In *ACM SIGGRAPH 2006 Papers*, pages 614–623. 2006.

[9] A. Naz. 3d interactive pictorial maps. Master's thesis, Texas A&M University: Available electronically from http://oaktrust.library.tamu.edu/handle/1969.1/1571, 2004.

[10] Y. I. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, 2001.

[11] R. Pausch, J. Snoddy, R. Taylor, S. Watson, and E. Haseltine. Disney's aladdin: first steps toward storytelling in virtual reality. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 193–203, 1996.

[12] G. G. Rice III. Caricaturing buildings for effective visualization. Master's thesis, Texas A&M University: Available electronically from http://oaktrust.library.tamu.edu/handle/1969.1/3196, 2006.

[13] M. Saldana. An integrated approach to the procedural modeling of ancient cities and buildings. *Digital Scholarship in the Humanities*, 30(suppl_1):i148–i163, 2015.

[14] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes. A survey on procedural modelling for virtual worlds. In *Computer Graphics Forum*, volume 33, pages 31–50. Wiley Online Library, 2014.

[15] M. Subramanian and E. Akleman. A painterly rendering approach to create still-life paintings with dynamic lighting. In *ACM SIGGRAPH 2020 Posters*, pages 1–2. 2020.

[16] J. van Muijden. Gpu-based procedural placement in horizon zero dawn. GDC, 2017.