Mass Scenes Rendering Framework

Dušan Bezák
dushan@ksp.sk

Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava / Slovakia

Abstract

The computer-based creation of realistic images often requires to generate large amount of similar objects – a mass scene. Common modelling and rendering tools offer only the support to copy or clone a specific object. This paper suggests an alternative approach. By providing a parametric description of the models, it is possible to generate many different objects (instances) automatically. The second step is to put these objects into the virtual scene. Different techniques for automatic object positioning (layout) are proposed. The results are tested on the Persistence of Vision Ray Tracer (POV-Ray) platform. To fulfil the desired tasks, additional commands for the POV-Ray scene description language are created.

Keywords: mass scenes, photo-realistic rendering, parametric objects, layout, POV-Ray.

1. Introduction

One of the goals of current computer graphics is to produce images with as much realism as possible. To achieve this, there is not only the need to improve old (or invent new) rendering techniques, but also to enhance the scene. In the recent years hard work has been done to make the modeling tools as comfortable and easy to use as possible. An experienced graphic designer is capable of creating a 3D model of almost any shape he can imagine. However, what good is an excellent three-dimensional model if it is in the scene alone? To create a realistic-looking scene it is necessary to place many entities at proper places. It is the scene composition that often leaves in the spectator the biggest impression. A brief glance at what support is available to the scene designer to make a scene with many objects constitutes an opinion that a mass-scene creation and rendering tool would improve both the quality and the comfort of a 3D-scene design.

1.1 Typical mass scene

Typical scenes that have been considered during the design of the mass-scenes creation tool include: characters in an audience, a flock of birds, a lawn with blades of grass, trees in a forest, a hairy monster, stalagmites in a cave, cars on a parking place.

If the scene author has enough time and patience, he might be able to do a very nice mass-scene without any special instrument, but with a great effort. Current modelling tools provide only two techniques to clone an object: instantiating and referencing the particular object. Although they are quite good to model small quantities of objects, they are not sufficient for mass scenes. The aim of this paper is to make the scene creation work much easier – what can be made by the computer should be done by the computer without any painful, long lasting or boring human assistance.

1.2 Instantiating an object

Instantiating is one of the basic operations that are in almost all software products. It is often called copy & paste. This facility makes possible to create a lot of instances, to place them into the scene and later change their properties. Some modeling tools provide functions to make the placement automatically – in a row, on the circle, or even some more sophisticated layouts.

1.3 Referencing an object

Referencing is almost the same as instantiating. The only difference is that if an object is cloned as a reference, after making a change in the original object, all the references are changed too. This has a big advantage if there are changes to be made after the objects have been cloned. E.g. the author forgot to add eyes to a creature that has already been cloned hundred times. After adding the eyes to the master (original) object, all the slaves (references) receive their own eyes immediately.

These two techniques have one main disadvantage – all the changes to the clones are to be done by the designer. The computer only makes the clones that are exactly the same as the original. However, for a realistic look the instances must differ in some details.

Due to the lack of time or patience, the author of the scene alters only a scarce number of the instances. At the first glance the result seems to be quite nice and fulfilling the intention, but after a short observation the objects seem to be somehow periodical and disturbingly regular, the scene is not as realistic as should be.

1.4 Special modeling techniques

A lot of hard work has been devoted in the recent years to some special types of mass scenes and parametric descriptions of objects. The results have usually the same basic scheme – the object to be modeled is considered from two points of view, the first is how an instance of the object can look like, and the second is how the instance behaves in an area or as a part of a whole entity. Then a tool is developed that confirms the assumptions. An example of such paper is [1] where the authors studied plant ecosystems and then rendered scenes consisting of thousands of generated plants in amazing photo-realistic quality. Another example is the modeling of human hair [2] by making a physical model of human hair, and thus generating the whole head.

Also the movie producers have their own special modelling tools that are many times developed exclusively for a desired type of scene. For instance the famous company Pixar developed in 1998 for the movie *A Bug's Life* new methods for modelling and animating large crowds of figures, but they keep their technical information in secret.

Nevertheless, all these modeling techniques are quite specialized for a particular type of scene. This paper introduces a technique that is not dependent on the scene type; it is a tool that is applicable on most mass scenes. On the other side it cannot compete (in the image quality) with a very special modeling tool, but this small handicap is counterbalanced by the generality of the approach.

2. Solution

To fulfill the planned, an environment that permits the creation of perfect 3D models is necessary.

The idea is to change the standard *clone-modify* paradigm to a new one: *parametrically describe*. The scene designer really needs only to provide the parametric information about the object. Then the computer is able to automatically generate clones of the object according to that parametric information. This scheme enables to create truly mass scenes with vast number of similar objects. Still no two among them will be identical.

The data describing the models should be extensible to keep the information including what and how could be modified in the cloned model, what are the parameters that the model must comply. The scene composition should be easily regulated. The environment should provide fast and acceptable graphic output. Moreover, there is the demand for a cheap availability of the environment. With these requirements on mind, POV-Ray seems to be an ideal platform for our purposes.

2.1 POV-Ray platform

Because the Persistence Of Vision Ray tracer (POV-Ray) is a source-free application, it cannot be cheaper and it cannot have better availability. For modeling it uses the POV-Ray Scene Description Language [3] that is quite general and allows to simply add new attributes, properties, directives, commands or functions. Binaries and sources of POV-Ray can be found and downloaded for free from the Internet [4]. Nevertheless the quality of POV-Ray's graphic output is comparable to the best commercial renderers.

The mass scene creation extensions of the standard scene description language should be divided into two categories:

- One object creation – parametric description of a 3D model.
- Scene composition – the layout of the objects into the scene.

For the first category one helper function (*makevalue*) and one directive (*alternative*) have been designed, for the scene composition a new adjustable command (*layout*) is introduced.

3. Parametric object modifications

The basic support for parametric object description comes from the standard POV-Ray's scene description language. The workflow of POV-Ray image creation is: create the scene source file (a text file), run the parser (that creates internal rendering structures – the scene), start the rendering engine. This allows the straightforward design of parametric objects using macros. A macro represents the objects to be generated many times in the scene. To let them all have different color, the only task is to add the color choice to the macro. The object's macro is parsed several times and each time a different color is chosen.

3.1 Random function

The production of large amounts of different objects cannot be done without a good random function. In POV-Ray there is a random function that generates a random number in the range between zero and one with constant density function; that means each number has the same probability to be chosen. In the real life no measurable parameter of almost any object has such distribution [5]. The basic distribution is the so-called normal distribution, where the density function is the gaussian curve. If the parametric definition of a 3D character requires to generate a number representing the figure height, it is much better to use a random number generated with the normal distribution instead of the standard random function. Maybe there is no reason to do such (a bit more complicated) calculation during the generation of only one instance of the object, but it is a real requirement for mass scenes.

What if the scene author wants to have a few small persons and a lot of tall ones? The normal distribution cannot be used. Still there is no reason why the author should not enter the density function that satisfies his intention. That is exactly the first extension of the POV-Ray scene description language – a function that generates a random value according to a given density function and its placement (the centre and the dispersion).

To enter the data describing the density function into the POV-Ray scene description language, an array comes to good use. The values in the array represent the density function ordinates, the abscises are equidistant. Two examples are given – a constant density function, and a gaussian-like density function:

```
#declare constant_density = array[2] {1.0, 1.0}
#declare gaussian_density = array[9]
    {0.0, 0.1, 0.5, 0.9, 1.0, 0.9, 0.5, 0.1, 0.0}
```

The code requesting a random value having the gaussian density function (according to the above example) applied to the height of a human – centred to 170 centimetres and with the dispersion of 30 cm will look similar to this one:

```
#declare height = makevalue(gaussian_density,
    170.0, 30.0, random_stream);
```

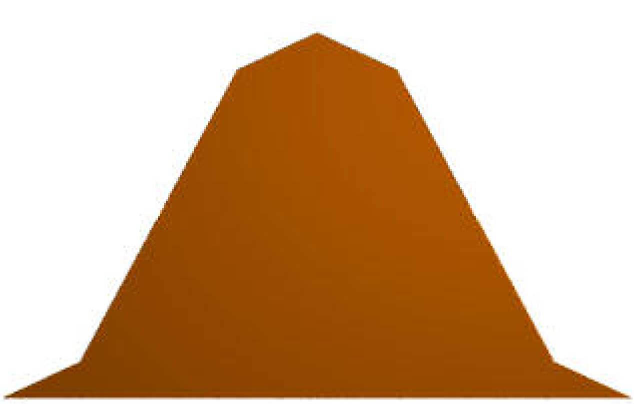

Figure 1: The gaussian density function data.

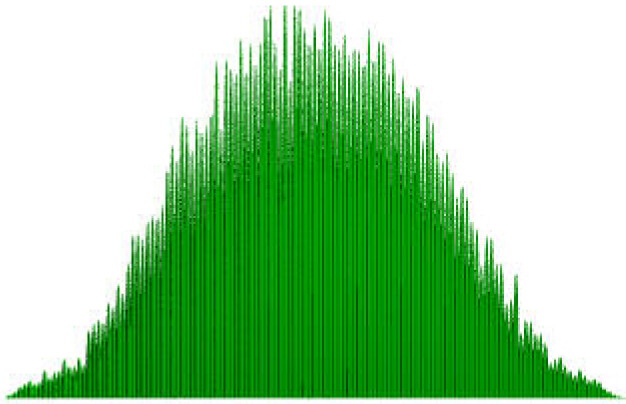

Figure 2: The makevalue function empirical data.

Where could the generated numbers be used? Anywhere – for sizes of the objects, for color values, for position coordinates, for pattern modifications, anywhere where a random value is useful.

3.2 Alternative

The second randomization of the macro representing the parametric object comes from the need to make decisions. Suppose, (for instance) that it is needed to decide whether a person will wear short or long trousers. Perhaps it is possible to describe the trousers length by the makevalue function with proper settings, but if there is the need to make totally different objects for the trousers, it has to be determined which type of the trousers to use. An alternative directive is the second contribution to the POV-Ray scene description language.

Usually the scene designer has the feeling of the percentage – the partial amounts of how many of the generated objects should comply a given condition. In the above example it could be stated that (for example) sixty percent of the people would have long trousers, and forty percent the short ones. An alternative directive added into the macro could express it in this way:

```
#alternative (random_value)
    #case (0.60) longTrousersMacro()
    #case (0.40) shortTrousersMacro()
#end
```

The alternative could be used wherever a decision has to be applied – for color selection, for the choice of object type, to choose any optional orientation. It is advantageous to use the alternative in the macro of the generated object to determine which object should be actually generated. A typical example occurs while modelling a common food – a letter soup, where the soup contains different soup elements:

```
#macro soupElement()
    #alternative (random_value)
        #case (0.86) letter()
        #case (0.08) carrot()
        #case (0.06) parsley()
    #end
#end
```

4. Layout techniques

A lot of mutually different objects could be generated. The task is to put them into the scene – to find the proper reference points where the generated objects will be placed. Three different techniques have been developed and are introduced in this paper. The techniques are dependent on the dimension of the layout problem. The basic three-dimensional problem means that the reference points should be distributed in some pre-defined space. How to describe the given 3D space? The POV-Ray environment gives the answer straightforward: by a 3D object described by the scene description language.

The same idea should be applied to a two-and-half-dimensional problem – to find the reference points on top of a terrain. The description of the terrain should be again any standard 3D object.

Usually (in mathematics) a two-dimensional problem is easier than a more dimensional. In the layout task, the two-dimensional problem means to find the reference points on the surface of any 3D object. It is quite hard indeed, because the 3D models are not described as standard meshes, but they can be any CSG compositions of non-elementary primitives.

4.1 Layout inside an object

Given are a 3D object (to become the skeleton for the layout – *the layout object*) and a parametrical description of the object to be many times generated (the object's macro – *the macro object*). The task is to find the so-called *reference points* in the layout object to place the objects generated by the macro. The ray-tracing capability of POV-Ray gives the opportunity to call a function *inside* to ask if a (3D) vertex is inside the layout object. The algorithm is quite straightforward: generate random vertices inside the bounding-box of the layout object, for each randomly generated vertex ask if it is inside the layout object and if the test succeeds, a reference point has been found.

Figure 3: An example of the layout inside technique. The layout object is a conical object representing the soup. The objects in the soup are automatically generated by the soupElement macro described above.

4.2 Layout on top of an object

The input data specification is the same as in the layout inside, but now the generated reference points should be on the top of the layout objects. This is the most common layout request – the basic association when talking about mass scenes is a crowd of people. And the people generally stand on a ground. During the scene preparation the author knows the exact 3D model of the layout object (e.g. a street, a forest terrain, etc.) where he wants to have an automatically generated crowd.

Again the ray-tracing capabilities are used. The rays are not fired as usually (from the eye through the projection plane), but they start somewhere above the layout object, and they are heading directly downwards. If the ray hits the terrain object, a reference point has been found to place an object (to use the macro to generate an object automatically).

To summarize the algorithm: find the bounding-box of the layout object, generate random vertices in the top face of the bounding-box, for each of these randomly generated vertices make a ray directed downwards (in the proper coordinates), use the ray-casting method for the layout object. If a hit occurred the reference point has been found, otherwise repeat the process again.

Figure 4: Characters generated on the terrain using the layout on top technique.

4.3 Layout on the surface of an object

The input data specification is the same again – the macro of the object to be generated many times, and the object that will serve as the skeleton of the layout. The task is to find a random vertex on the surface of the layout object with constant density function. The constant density function requirement is very important, because there is the need of uniform distribution of the generated reference points on the layout object's surface. Why? A typical surface mass scene – a hairy monster, could answer this. It is expected that the density of the hairs of the monster is the same everywhere on its body. If the reference point's distribution were not uniform, there would occur some areas with higher hairs presence.

An algorithm to find the random reference points on an object's surface is again based on the ray-casting technique. Two random vertices on the surface of a bounding sphere are generated, their connection constitutes a ray (that is actually fired from the first vertex heading towards the second), and the first intersection of the ray with the layout object is taken as a reference point. The reference points do not have necessarily the uniformity property that has been demanded (in cases the layout object is not a sphere), but it works quite fast. To obtain the uniform reference points distribution, a simple check for the relative reference point distances can be applied – if a ray in the algorithm has hit a vertex, it becomes a reference point only if the distance to all other reference points is greater than a given value.

A big disadvantage of this method is that if the layout object has too concave surface (e.g. a vase) the probability of hitting a vertex on the inner surface of the object is too small. In most situations, this is not a serious drawback because the concave parts of objects have usually limited visibility (if it is hard for the rays to get into the vase, it is also hard to see there). However, to solve this handicap another method has been developed.

The second algorithm gives better results but is (much) slower. It finds the reference points on the surface of any object (or almost any – except some fractal-based surfaces) with uniform distribution. The idea is that random vertices are generated in the bounding-box of the layout object. For each of these vertices a very small (smaller than the smallest bend of the layout object's surface) sphere is considered. On the surface of this sphere a random vertex is chosen (called s; let the sphere centre be called c). A ray is fired starting from the vertex s, targeting the centre c. If the ray hits the layout object's surface on the line between s and c, the vertex that was hit is a reference point. This algorithm provides reference points with uniform distribution on the surface of the layout object.

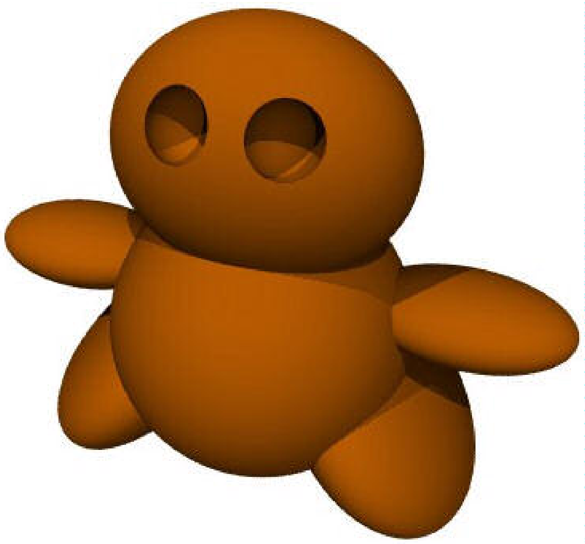

Figure 5: The layout object.

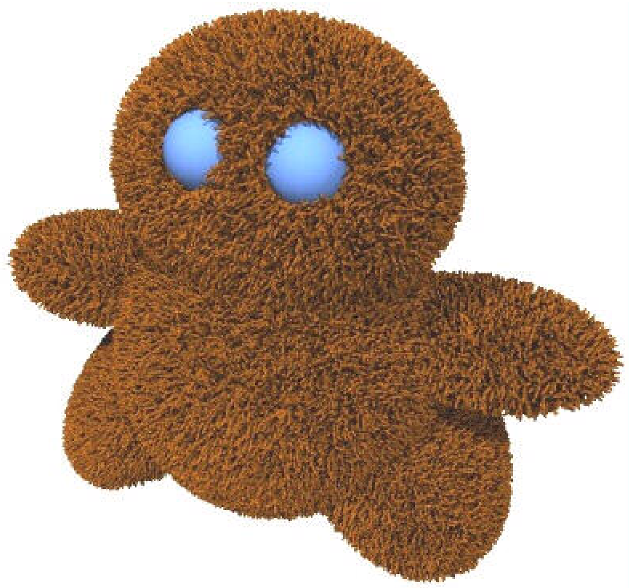

Figure 6: A hairy monster. The result of the layout on the surface technique.

4.4 Layout parameters

After a reference point is generated, the object creation macro is called to create the object to be placed at that reference point. There is nothing that prevents the macro objects from mutual overlapping. An instrument that forbids the overlapping is the minimum distance parameter. If a minimum distance is provided, for each new reference point that is generated, the distance to the nearest reference point is taken (it is the minimum distance to all reference points) – and it is checked if it is greater than the given value. If not, the reference point is discarded and the respective generation algorithm continues. After a given number of unsuccessful retries the algorithm stops with the output that no more reference points could be found.

Another parameterization is to allow the object's macro to be called with additional environment describing parameters. The macro has the possibility to alter the generated objects depending on

- the location in the scene (the reference point). The macro object is able (for instance) to change its color or size according to the location. If the scene is a crowd of people, the persons (or their heads) could be rotated according to their locations to look at some special place (e.g. the audience looking at the ball in a tennis match).

- the ground orientation (the normal of the ground). This parameter describes the elevation of the ground at the reference point. It is useful (for instance) to rotate the hairs of a monster so that the roots of the hairs are perpendicular to the layout object's surface.
- the color of the layout object at the reference point. This is very useful to generate the macro objects with different behavior at different places in the scene. Typical example is a meadow with flowers, where the flower-types are chosen according to the texture of the meadow base object – if the texture is some gradient transition, one side of the meadow will have a majority of one type of flowers, the other side will have flowers of the second type. It is also possible to alter the height of the grass blades according to a well-chosen texture.

Also one important output parameter has been developed: The macro could output a boolean value determining if it wants to be generated (with the proposed settings) or not. For instance in a scene with characters on a terrain it is not very common to have a character on a high slope where the elevation is too big. The object's macro can check the input parameters and decide that the proposed reference point is not very suitable for the macro object to be generated there. It outputs false and the reference point generation continues with another try.

5. Complexity and Time Issues

When talking about mass scenes the computational complexity is very important. The most relevant is the time complexity. The time necessary to create a nice image can be divided into three exclusive parts: the time to prepare and to design the scene, time for the parser to read the source and to build the internal structures, and the time that the rendering engine needs to render the image.

The time needed for the scene composition has been reduced significantly. The scene composer does not have to mark for each of the generated objects the place where to put it. The boring long lasting work surely belongs to the computer.

Execution of all proposed commands takes place in the parse time. Although the parse time is much smaller that the other two times, the shortening of that time is still not a worthless job. During the scene preparation the designer needs to preview the image (at low resolutions) many times, where a longer parse time is very annoying. To accelerate the layout minimal distance comparison, some types of hash-tables are applicable. As the reader has certainly noticed, all the layout techniques work this way: generate a random entity (a vertex or a ray) and try to use the entity to find a reference point. If the reference point has been found, it is good, but if not (the vertex has not been inside or the ray did not hit anything), the quest for a reference point has to be repeated. To avoid these unlucky choices, some space partition trees could be made that would assign correct probabilities to the tree nodes. Thereby a random entity (the vertex or the ray) will have lower probabilities to be generated in the regions where the chance to find a reference point is lower.

The render time acceleration is a hard task and out of the scope of this paper. There is one suggestion how to save some render time (and quite a lot of memory): to count the distance from the camera to the object being generated in the object's macro to reduce the model quality according to this distance. It is sufficient if the models in the front (near the camera) have higher detail, and the others at the back have lower detail. However POV-Ray makes for the rendering some sort of scene object trees, thereby the render time rises by the logarithm of the scene complexity – the number of objects in the scene. In other words it almost does not matter if there is one thousand of objects in the scene or ten thousands.

As we have chosen POV-Ray, it works on almost any platform and it is possible to accelerate the parse and render times by a migration to a faster machine.

Scene	# of elements	Parse Time	Render Time
The letter soup	1500	0:00:05	0:01:35
The characters on a terrain	50	0:00:01	0:00:05
The hairy monster	45000	0:04:08	0:01:03

Table 1: The parse and render time comparison, executed on an AMD K6-2 350 with a preview resolution (320 x 200 pixels) without the antialiasing.

6. Conclusions and Future Work

This paper has introduced a general technique that allows to create and to render mass scenes. It is impossible to make something like a mathematical proof of completeness (that it can be used with all mass scenes), but the technique is at least helpful in the process of mass scene creation. It has been tested with many various scene types, and it showed to be very easy to use and conductive.

It must be said that the proposed layout techniques do not consider any interaction between the placed objects. An instance is a crowd scene where a figure holds one hand of another figure. This small interaction can be avoided by making not only one-character macro objects for the layout, but also a couple should be a macro object. There are still some scenes (like soap bubbles on the water) that need a real interaction between the objects (bubbles) to modify the object shape. To cover these scenes a small modification could be made – the objects will not be really created in the layout process, but only the reference points will be placed into some array and the object creation macro will have the possibility to use this array, to look around and see where and what is around. E.g. a soap bubble will know its surroundings and according to them will modify its shape.

The future work includes the creation of additional layout parameters according to the requirements of some special mass scenes, where the set of current parameters would be insufficient. To achieve some state of completeness maybe also the opportunity to create regular layouts – multidimensional grids or rows should be added. Another research path should be the possibility to create truly dynamic animations.

This mass scenes rendering framework is not limited to the POV-Ray platform. It can be easily ported to any other 3D environment, where the adding of some functionality is possible (e.g. in 3DStudio the possibility to make own plug-ins).

7. References

[1] Deussen, O., Hanrahan, P., Lintermann, B., M ch, R., Pharr, M., Prusinkiewsicz, P.: Realistic modeling and rendering of plant ecosystems, in *Proc. of Conference SIGGRAPH*, pp. 275-286, 1998.

[2] Anjyo, K., Usami, Y., Kurihara, T.: A Simple Method for Extracting the Natural Beauty of Hair, in *Proc. of Conference SIGGRAPH*, pp. 111-120, 1992.

[3] Persistence of Vision Development Team: POV-Ray Scene Description Language, in *POV-Help documentation* at *http://www.povray.org/*.

[4] Anger, S., Bayer, D., Cason, C., Dailey C., Demlow, S., Enzmann, A., Farmer, D., Wegner, T., Young, C.: Persistence of Vision Raytracer, at *http://www.povray.org/*.

[5] Riečanová, Z., Riečan, B., Olejček, V.: Numerické metódy a matematická štatistika, Numerical Methods and Mathematical Statistics, ALFA, Bratislava, 1983.

8. Examples

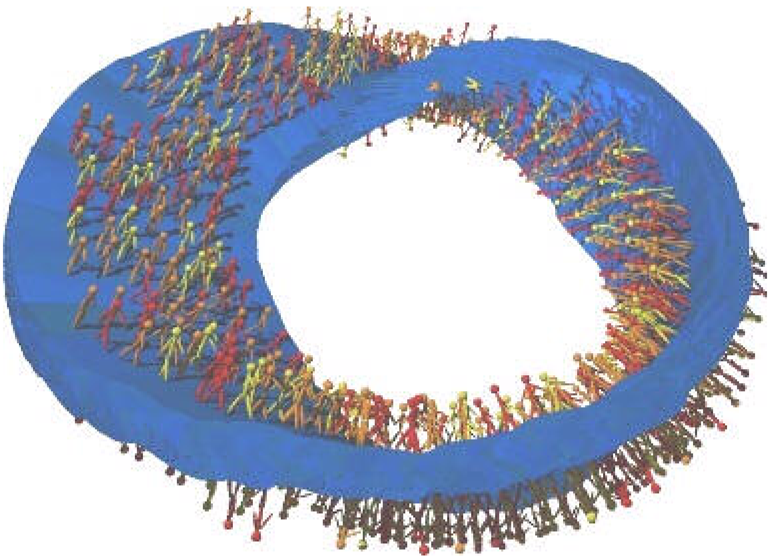

Figure 7: The layout on top technique applied to a Mobius strip.

Figure 8: Dwarves Mass Scene. Created using automatic layout of dwarves and stalagmites.