EUROGRAPHICS 2007 / D. Cohen-Or and P. Slavík
(Guest Editors)

*Volume 26 (2007), Number 3*

# Multiresolution Modeling Using Fractal Image Compression Techniques

paper1193

**Abstract**

*This work presents a new approach to the multiresolution modeling of polygonal meshes. This approach is based on the theoretically well-established fractal image compression techniques. A polygonal mesh is represented as a fractal using an iterated function system (IFS). In this way, a level of detail can be obtained over a region of the mesh by successively iterating the IFS. The main advantage is that it becomes possible to recover new levels of detail that were not present in the original mesh, so that the quality is not lost as the observer approaches the mesh. Another characteristic is that the same representation can be used over textures, and in this case the algorithm is directly implemented over the GPU. The visualization time obtained allows this new approach to be used in real-time interactive computer graphic applications.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; I.3.7 [Computer Graphics]: Fractals.

## 1. Introduction

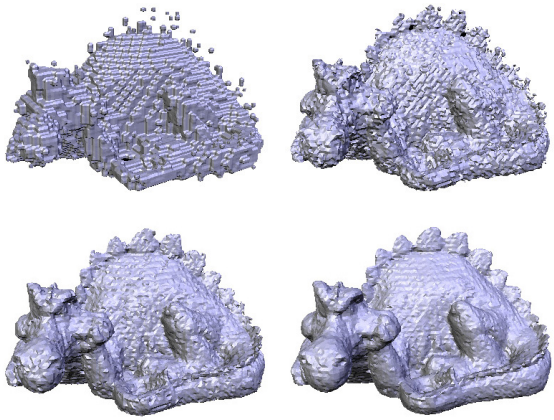


**Figure 1:** *The first four decompression passes of the Phlegmatic Dragon 3D model as a fractal.*

Interactive visualization of large triangle meshes is a common problem that is present in engineering (CAD model visualization), architecture (virtual walkthroughs), GIS (terrain model visualization), computer games, film post-production, medicine and so on.

Multiresolution modeling has proven to be a powerful tool for avoiding this problem, as shown by the large number of papers published on this research topic; these works deal with issues ranging from terrain polygonal meshes [Hop97, PFMC04, DWS*97, HDJ05] to general polygonal meshes [XV96, Pri00, ESAV99, FPM96, FMP98, ZC06, LE97, EMWVB01, Paj01, PD04, GH97, DP02, KL01, KL03, JKK04].

In this work a new approach to multiresolution modeling of polygonal meshes is presented. This approach is based on the theoretically well-established fractal image compression techniques. This new method codifies a multiresolution model as a fractal using an iterated function system (IFS) representation. The view-dependent levels of detail are recovered by iterating over the IFS until the required resolution is reached. Since the models are codified as a fractal it is possible to recovered a resolution with more detail than that existing in the original model. This new detail is not an artificial one as in previous works [PFMC04], but is based on the self similarity of fractals. Moreover, the codified multiresolution model sizes are smaller than the original polygonal model.

Unlike other multiresolution models that have previously

been published, our multiresolution model does not need a simplification or a decimation algorithm to obtain the set of levels of detail that the model stores.

The decoding algorithm can be easily coded as a GPU pixel shader. The same algorithm used to represent the polygonal mesh can therefore be used to represent a texture having the same properties. In particular, new self-similar detail can be added to the image if the observer approaches the surface. Obviously, implementing the decoding algorithm in the GPU frees this task from the CPU.

The rest of this paper is organized as follows. Previous work on multiresolution modelling is reviewed in Section 2. Section 3 presents the theoretical background to fractal image compression and how this is used to build a multiresolution model over a height field terrain model. This section also shows how the model is extended to codified polygonal meshes. The description of the experiments and their results are given in Section 4. Finally, conclusions and future work are discussed in Section 5.

## 2. Related Work

In [XV96] Xia et al. presented a view-dependent multiresolution model for polygonal models based on a hierarchical-vertex structure named *merge trees* developed by the authors. Each *merge tree* is built bottom-up, from the high detail mesh to a low detail mesh, using edge collapse. An edge collapse establishes a parent-child relationship between the two vertices of the edge, the vertex that remains being the parent of the vertex that collapses over it. During the visualization process a front through over the tree with the visible vertices is maintained, and this front through is incrementally updated by taking advantage of frame-to-frame coherence.

In [Hop97], Hoppe independently presented a view-dependent multiresolution terrain model also using edge collapses. Prince [Pri00] extends the model to general polygonal models.

In [ESAV99], El-Sana presented a view-dependent multiresolution model that uses topology simplification through *virtual-edge* collapse. A *virtual-edge* connects two vertices in which Voronoi cells share a Voronoi face and are not edges of the model. In contrast to [XV96, Hop97], in this case implicit dependencies are used, so fold-overs are more easily checked as the multiresolution model is updated.

The multiresolution model presented by De Floriani et al. [FPM96, FMP98] uses a *directed acyclic graph* (DAG) as the data structure to build a view-dependent multiresolution model. Each node of the graph stores local modifications, which are typically vertex insertion or removal. A cut over the DAG determines which triangles to visualize. Traversing the DAG in order to obtain a valid cut has a high computational cost. Zheng et al. [ZC06] analyze some strategies with which to traverse the DAG in an efficient manner.

The hierarchical structure used by Luebke et al. [LE97] is built using a generalization of the vertex collapse and split over an octree. In each simplification step, all vertices within an octree cell are replaced by a representative vertex. During the visualization process, a cut over the tree is obtained and it is incrementally updated by taking advantage of frame-to-frame coherence.

In [EMWVB01], Erikson et al. presented a multiresolution model based on a hierarchy of levels of detail (HLODs). Inner nodes of the hierarchy represent individual simplified parts of the original polygonal model (LOD nodes) as groups of these parts that are simplified together (HLODs nodes). This model allows some limited edition by recalculating, during the editing process, the parts of the hierarchy that have been modified.

In [Paj01, PD04], Pajarola et al. presented a hierarchical multiresolution model based on a half-edge representation of the polygonal mesh. The hierarchy is built using a modified version of Garland's simplification algorithm based on edge contractions [GH97], in such a way that the topology of the mesh is not preserved during simplification. To minimize the dependency between vertices in successive contractions, candidate edges for simplification are selected in mesh zones that are a long way away from each other. In [DP02] this model is extended to work out-of-core.

In [KL01, KL03], Kim and Lee presented a new scheme to build view-dependent hierarchical multiresolution models based on the dual space of the polygonal model, which the authors called *transitive mesh space of a progressive mesh*. A *dual piece* over the dual space is assigned to each vertex in such a way that it does not intersect any other *dual piece*. Edge contraction is used to build the hierarchy and the *dual piece* of the vertex that remains after each contraction is the union of the *dual pieces* of the original vertices. There is no dependency between the *dual pieces* of the hierarchy which have no parent-child relationship and, hence, very drastic changes can be obtained in the simplified mesh. In [JKK04] the authors extend the model in order to transmit it through a communication line.

To avoid the large amount of consistency calculus needed in [KL01, KL03] as the model is simplified and also to obtain a better control over the error, in [Gum05] Gumhold presented a multiresolution model based on *dual pieces* which uses vertex removal to build the hierarchy; in this way the triangles are stored in the tree nodes instead of vertices.

M. Duchaineau et al. [DWS*97] presented the ROAMing Terrain model. This view-dependent multiresolution model for terrain models originally used bintree triangulation as data structure. Later, in [HDJ05] the structure was changed to a diamond data one. The diamonds or triangles are dynamicaly updated using split and merge operations. To optimize these operations two priority queues are used, the split queue contains all triangles for a given triangulation, while the merge queue contains all the mergeable diamonds for

a given triangulation. As the model chages, split and merge operations are respectivelly done over the diamonds or triangles in the queues. Another optimization used in the model is incremental stripification of the triangles before rendering. The authors have extended their model to work on arbitrary geometry, and improve it to manage texture slip and merge, geometry and texture paging, occlusion culling and wavelet compression.

## 3. Model Description

The multiresolution model presented in this paper is based exclusively on fractal image compression techniques. This section shows the basis of fractal image compression as an IFS and how it is possible to build a multiresolution model from a digital terrain model (2.5D). Then it is shown how these ideas can be extended in the case of 3D models.

### 3.1. Fractal image compression

Fractal image compression is based on the fact that images, like fractals, are redundant in the sense that they can be built from transformed copies of themselves [Jac90]. In the same way that a fractal can be represented by a set of Iterated Function Systems [Hut81], an image can also be represented by a set of IFS [Jac90, Bar93, Fis92, FBMK95].

An IFS is a dynamic system $W$ of $n$ contractive transformations $w_i$ over a metric space. A 2D IFS maps the plane $\mathbb{R}^2$ over itself, and it converges as the system is iterated.

$$W(x) = \bigcup_{i=1}^{n} w_i(x) \quad \{w_i : \mathbb{R}^2 \to \mathbb{R}^2 | i = 1, ..., n\}$$

The map $W$ serves our purposes as a fractal scheme that codifies an image $I$ in 2D as a set of contractive transformations over itself. The fixed point of $W$ is an approximation to the initial image $I$.

**Encoding (compression):** Fractal encoding of an image $I$ finds the set of transformations $w_i$ that generates the image $I$ as they are iterated. If the elements of $\mathbb{R}^2$ in the original space are called domains $D$ and the elements $\mathbb{R}^2$ in the destination space are named ranges $R$, then encoding consists in finding the domain $D$ that, when transformed, best fits each range $R$ in accordance with the metric that has been defined. The *fixed point theorem* for contractive mappings imposes that the set of transformations be contractive in such a way that the iteration over the IFS converges to the image $I$. On the other hand, images are not exactly auto-similar and so, in order to obtain a better fit between domains and ranges, brightness $o$ and contrast $s$ factors are introduced into each transformation, taking $s < 1$. By so doing, the search process must find the optimum $o$ and $s$ that transform $D$ while minimizing its difference with $R$ under the defined metric.

An IFS allows us to choose the sizes and arrangements of range $R$, and also the precision of the transformation parameters like $\mathbb{R}^2$ or the $o$ and $s$ factors. This makes it possible to choose the size needed to store an IFS. Generally, this size is less than the size of the original image $I$ and some information is lost. The encoded process is a lossy compression process.

**Partitioning:** An important issue, directly related with the quality of the codified image, is how ranges are chosen. If we impose that ranges cannot be superposed, then they can be chosen over a grid of constant size. In this way, the quality of the codified image is directly dependent on the grid size, since the more transformation the IFS stores, the better the quality of the codified image will be. Ranges can be chosen adaptively, in such a way that the size of the grid can be adjusted to the details of the image, taking larger sizes of the grid in those regions of the image with little detail, and a smaller grid size in those regions of the image with more detail. Adaptive grids diminish the size of the codified image while maintaining its quality. *Quadtree*, *HV*, and triangular subdivisions are the most frequently used types of adaptive partitioning [Fis92, FBMK95].

**Domain set:** A domain of the image can be any region of the image and all of its orientations and flips. The size of the domains must be bigger than the size of the ranges in order for an IFS to be contractive. On the other hand, one domain can superpose onto others. For the sake of simplicity, domain sizes are chosen with double the size of the range sizes, thus making it possible to perform an average subsampling so that a domain can be compared with a range. For a given image of size $n * m$ and domain size $t * t$ there are $(n-t+1) * (m-t+1)$ different domains. Contracting and comparing each of these domains with each range in order to find the domain that best fits a range has a very high computational cost. To reduce computational time domains can be chosen either over a regular grid or at random. But a better strategy that is commonly used [FBMK95] is to group together those domains that look more like each other, and compare each range only with the domains in the group that resemble each other.

**Decoding (decompression):** The compressed image can be recovered by simply iterating the IFS over a *seed* image.

$$I \simeq |W| = \lim_{x \to \infty} W^{on}(x)$$

The result of decoding is independent of the seed image as $x \to \infty$ but can cause the fractal to converge in more or fewer iterations. The more the seed image looks like the decoded image, the faster the IFS converges. Decompression can be performed in several ways:

- Recovery with fixed resolution: for obtaining a decompressed image of a specified size. An IFS is iterated over a seed image, the decompressed image will have the same size as the seed image. This is the direct method and yields

the best quality results of the decompressed image $I$. Nevertheless, seed images of sizes other than those of the encoded images can be used. IFS iterates by subsampling the pixels of the images. Hence, a decompressed image of arbitrary size can be obtained.

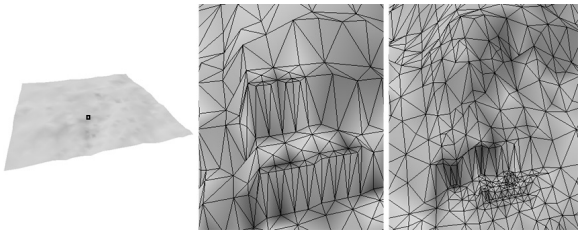
- Recovery with continuous resolution: for recovering the value of the image at a given point independently of the rest of the image points. In this case, the iteration process must be inverted, that is, by beginning with the real coordinate of the final point we are interested in, the set of transformations $w_i$ is found. This process is repeated a number of times and for the $s$ and $o$ factors of all transformations $w_i$ involved. In the end, the value of the seed image is taken and the transformations are performed in an inverse order to the one in which they were found. The final result is the value of the IFS iterated only over the selected point.

**Properties:** An image codified as an IFS has the following characteristics:

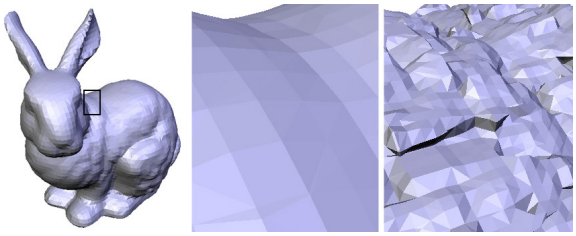
- Selective quality: in the moment of building the IFS one area of the image can have more priority than another and, thus, regions of interest (ROI) are selected during the compression of $I$. In order to use ROIs, some parts of the original image can be split with a large number of ranges and also a large number of domains are used while obtaining the transformations $w_i$. Thus, the ROIs will have a better resolution while decompressing the image.
- Infinite resolution: There is no limit in the size a fractal is decompress. An image codified as a fractal can be decodified at every resolution, much more, much less or at the same size that the original one. The boundaries of the image codified are in the range $[0,1][0,1]$ and it is possible to recover information of the image for any given point within this range. This gives an added detail to the image as shown in Figure 2(a).
- Continuous range of colour: although the original image had a finite number of colours only, the codified image as an IFS has a continuous range of colours between $[0,1]$. This property is very useful if the colour resolution in the image representation needs to be changed.
- Progressive refinement: The more iterations over the IFS the more the quality of the image. The image obtained after an iteration is the image seed taken to perform the next iteration. This way the iterative process can be stopped after reaching some pre-specified error.
- Selective refinement: Since the decompression process is done one pixel every time, some regions of the image can be decompress at high resolution than others.

**Lossless compression:** Fractal compression is a lossy compression technique. Nevertheless, lossless or quasi-lossless compression can be obtained. One method to obtain lossless compression is to use an IFS variation with place-dependent probabilities [Bar02]. Another method is to take a regular partitioning of fixed size with range size of one

(a) 2D colour fractal.



(b) 2.5D digital elevation model.



(c) 3D model.

**Figure 2:** *Added detail on fractal decompression. The image on the left is a global vision of the model. In the middle there is a zoomed in region. The image on the right is the fractal model decompression on this zoomed in region.*

pixel. To retain the topological information in the comparison range-domain, the original image $I$ is supersampled in such a way that ranges of size two and domains of size four are obtained. This way one transformation by pixel must be stored, increasing the compress image size. Although the size of the compressed image could be bigger than the original one, sometimes is preferable to use a lossless fractal image.

**Colour image compression:** Fractal image compression techniques can be easily extended to images with more than one colour component. Fractal colour image compression means to codified independently each colour component of the image. In the same way, colour image decompression means to decodify all colour components of the image. The YIQ colour space (luminance, hue, saturation) is advisable when using colour images [FBMK95, KB96], because the human eye in not particularly sensitive to colour information, but more sensitive to brightness. In this way, it is possi-

ble to compress with a high ratio the I and Q canals without loss of quality. The IFS of the I and Q canals used to be reduced to a half or a quarter of the original image size, and they are compressed with a bitrate lower than that of the Y canal, so the bitrate of the overall colour image is reduced.

### 3.2. Fractal compression applied to elevation maps

This section shows how the fractal compression techniques described in Section 3.1 have to be applied in order to obtain a multiresolution representation of a digital elevation model (DEM from now on). DEM models are used to represent regular grid-sampled data of terrains. A DEM elevation map allows access to an elevation value using a pair of coordinates. These coordinates allows the height data to be arranged as a two-dimensional array. Each height of the DEM can be interpreted as a grayscale level. Thus, the DEM is similar to a grayscale image.

**Compression:** In order to set up the ranges we used a quadtree-based adaptive partitioning. The user chooses the maximum number of transformations for an IFS representation. Should this maximum number not be reached, the node that gives rise to the greatest error in approximation is splited, its transformation is erased from the list and a new transformation is added for each of its four child nodes. To speed up the process of minimizing the error transformation, the domains are grouped together by appearance in order to search between only the domains with a similar one. In our case, the appearance criterion is based on the domain's brightness arrangement. This works by subsampling the entire domain to a 2x2 size, classifying the new arrangement of values from lower to higher, and grouping the domain with the ones that share the same arrangement.

The metric we have chosen to compute the distance between two images $x_{i,j}$ and $y_{i,j}$ with size $t*u$ is the rms metric, which is defined by the following equation:

$$d_{rms}(x,y) = ||x-y||_2 = \sqrt{\frac{1}{t*u}\sum_{i,j=1}^{t,u}(x_{i,j}-y_{i,j})^2}$$

with this metric, the search for the contrast $s_i$ and brightness $o_i$ that minimizes the difference is reduced to solve a *least squares* problem [FBMK95]. For the terrain, we used a lossy compression with a small lossy factor to preserve much of the original elevation map appearance. The compression is an off-line process, and compression times for different elevation maps are shown in Table 1. Finally, the elevation model compression can also be adjusted with ROI zones.

**Decompression:** The terrain rendering algorithm is the one that defines what points we need elevation for. For the purposes of this paper, a general *bintree*-based algorithm was used. This algorithm generates a regular triangle hierarchy. However, any algorithm of any type of structure can be used.

There are two options in order to obtain the elevation for each triangle vertex. The first is to decompress the whole of the fractal DEM with discrete extraction of the IFS. In this case, we recommend subdividing the initial DEM and compressing each piece as a separate fractal. In this way, decompressing one point will not have a global decompression cost on the DEM. But it will only be expected to decompress the piece of DEM in which the point lies. In addition to this, a small version of the original elevation map can be stored with the IFS with no compression at all. Thus, the decompression will be faster if the seed for each point is taken directly from its projection onto this thumbnail. However, a *continuous extraction* of the IFS will adjust much better to the terrain representation necessities. Furthermore, the value of every point would be extracted independently and there is also the possibility of extracting values for coordinates where the DEM had no information due to the infinite resolution of the IFS. This would be very advisable for representing algorithms of the TIN terrain type. We will use decompression of continuous resolution because it is the one that best fits the properties of selective decompression, progressive refinement and added detail. Moreover, we will not store any thumbnails to work as the initial seed. The seed for each coordinate will be the midpoint between the minimum and the maximum elevation because this value will approximate equally to every possible elevation.

View-dependent multiresolution models compute their approximation to the terrain according to vision parameters. In a similar way, fractal DEM decompression can also take into account vision parameters to perform selective decompression. Triangle culling, terrain clipping, camera distance or screen projection of triangles can be used for a guided decompression (see Figure 3). The quality of a fractal DEM coordinate decompression will be determined by these parameters.

IFS-based DEMs can progressively refine their heights. Beginning with the seed, and thanks to the IFS iteration algorithm, it is possible to compute a more precise refinement of the decompression using the actual elevation. This has been impossible until now with a common DEM and implies the capacity of a time controlled decompression. Now it is possible to decompress in a limited time and use the result in a later decompression. This property will be greatly appreciated in cases where the response of the terrain representation is required in real time.

In most terrain representation methods, noise is commonly added to points that exceed the maximum DEM resolution. The new fractal DEM does not have this problem due to the fact that fractals do not have resolution limits. Its use will, by definition, give even higher resolutions than the ones the DEM originally has. As shown in Figure 2(b), this gives an added detail that is far better suited to the original terrain than only adding a random noise.

**GPU texture decompression:** Thanks to the new pro-

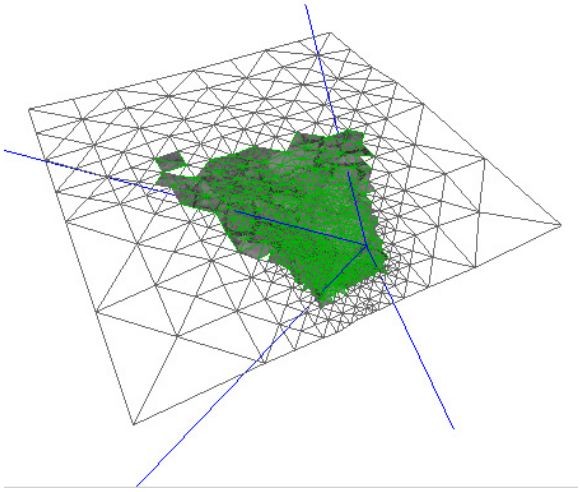


**Figure 3:** *Fractal terrain with viewport culling and selective decompression based on the camera distance to each vertex.*

grammable GPU technologies, terrain texture can also be a result of IFS decompression (see Figure 4). With the present limitations this IFS will have a regular structure with the same width and height in order to make the pixel shader feasible.

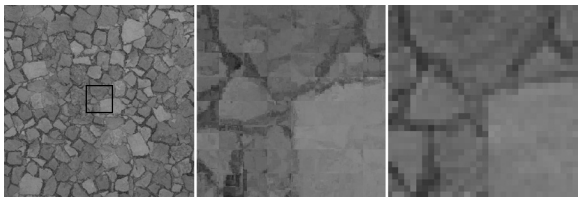


**Figure 4:** *Comparison between a polygon with the GPU fractal decompression (middle) and same polygon with a bitmapped texture (right). In the fractal texture no texels are visible.*

The pixel shader can receive the IFS description by encoding this in texture elements. In order to simplify, the IFS description is split into information groups and each texel of a texture will imply one range of the regular IFS. With process, four different textures are created. The first texture is named *texDOMPOS* and has information about the position of the domain in image coordinates. The second one is named *texSO* and defines the brightness $o$ and contrast $s$ values of each range. The third texture, *texORI*, is the last one with IFS information and encodes the domain's orientation. Finally, we recommend the creation of a fourth texture for shader enhancing reasons. This texture is an $8x4$ matrix which encodes orientation matrixes for each possible domain's orientation. It is also important to note that in many cases integer range textures must be recoded to real range ones. This is because graphic programmable shaders define texel values for their textures in the range $[0...1]$ for generalization purposes.

In addition to textures, certain variables will also need to be defined for the shader. The first two variables are dependent on the size of the IFS. These are *transx*, containing the number of ranges per side, and its inversed value, *invtransx*. Another variable will be the number of iterations of the decompression system of the IFS and it is called *iterations*. The last one will be *seed*, obviously consisting of the seed that the decompression begins with.

The pixel shader implements the inverse decompression algorithm for real coordinates. Its code is based on the following decomposition of the IFS iteration system:

$$
\begin{aligned}
f(0) &= seed \\
f(1) &= seed * s_0 + o_0 \\
f(2) &= (seed * s_0 * s_1) + (o_1 * s_0) + o_0 \\
f(3) &= (seed * s_0 * s_1 * s_2) + (o_2 * s_0 * s_1) + (o_1 * s_0) + o_0 \\
f(x) &= seed * factor0 + factorlast + factoracum
\end{aligned}
$$

following *factor0*, *factorlast* and *factoracum* the algorithm described on Figure 5.

```
factor0=s_0;
factorlast=o_0;
factoracum=0.0f;
for (i=1;i<n;i++)
{
  factoracum+=factorlast;
  factorlast=factor0*o_i;
  factor0*=s_i;
}
```

**Figure 5:** *Iteration algorithm for the fractal pixel shader.*

Thus, the entire pixel shader written using *GLSL* looks like Figure 6.

### 3.3. Fractal compression of polygonal meshes

This section explains how to extend the fractal image compression technique to polygonal 3D meshes. To do so, a volumetric representation of the mesh [Jon95] known as a *distance field* is computed to be used in its place. A *distance field* is equivalent to a grayscale 3D image in the same way an elevation map was defined as a grayscale 2D image in Section 3.2. This 3D image can be compressed and represented with an 3D IFS (IFS3D from now on). Once the IFS3D has been computed, the multiresolution representations can be obtained as approximations to the original

*distance field*. These approximations are simply decompressions of the IFS3D. Finally, a polygonal mesh can be reconstructed from the decompression of the *distance field* using the *marching cubes* [LC87] reconstruction algorithm.

**From 3D mesh to 3D image:** Commonly, a mesh is equivalent to a polygonal model (almost always a triangle model) in 3D. However, our method is not yet prepared to work directly on polygons. To be able to work with the mesh as a fractal, first it must be transformed into a 3D image.

The 3D image representation of the original mesh will be its *distance field*. A *distance field* is a 3D matrix (3D image) that is wrapped onto the mesh and stores the value of the distance between the mesh surface and the center of each voxel. The value of the distance is calculated in such a way that when the voxel is completely out of the mesh the value of the distance is equivalent to the maximum. On the other hand, when it is completely inside it will be equivalent to the minimum. And when the center of the voxel is placed on the surface itself the value will be zero. In our case, we codify the *distance field* as an signed byte precision 3D array in such a way that the values range between -128 and 127.

To construct the *distance field* we use an *octree* hierarchic structure. In this way, only the voxels which intersects the surface of the mesh are refined to the point of becoming the leaves of the tree. On the other hand, when a node does not intersect with the surface, the voxel that it represents and all its descendants voxels possesses the same distance.

The distance between the central point of the voxel and the surface of the mesh is computed as the minimal distance between this central point and any triangle of the object. It is possible to use a generic distance algorithm between 3D point and 3D triangle [Ebe99] to solve this problem. The same algorithm will serve to know if the central point of the voxel is inside or outside the mesh. If the normal on the triangle of minimal distance is facing toward the central point of the voxel it is said that the point is inside the mesh and a positive distance is assigned to it. In the opposite case it is said that the point is out of the mesh and a negative distance is assigned to it.

**Fractal compression of 3D Image:** The fractal compressionl of 3D images is a direct update of the case with 2D images. Although visually it is difficult to conceive an image in 3D, it is mathematically as feasible to apply the IFS method to 3D as to those in 2D.

To work in 3D first it will be necessary to consider the new ranges and domains that also happen to be in 3D. Another important change is the complexity of its homogeneous transformations that range from eight different possible orientations to 32. This implies that the set of possible domains increases in a critical way. Moreover, it is necessary to change the algorithm that minimizes the difference between ranges and domains. This is due to the fact that the number of pixels that they both contains changes. Otherwise, the compression process is carried out in the same way as in Section 3.1 and an IFS3D is obtained that is capable of compressing the *distance field* generated from the original polygonal mesh.

We have used an octree hierarchy so that to obtain the IFS3D it is only necessary to process those voxels that intersects the mesh. The number of transformations to be codified will depend on the necessities in each case. Nevertheless, considering ranges of a minimal size of two is enough for the mesh to have a visually acceptable quality. In contrast, if we require the mesh to have no loss in quality it will be necessary to use the lossless methods and this will considerably increase the final size of the compressed mesh. With regard to the domains to be borne in mind, as always the rule is: the more, the better. We advise considering all the possible ones although the compression time increases a lot. In order to avoid affecting the compression time in excess, dominions grouped by similarity can be used.

**Decompression:** The decompression of an IFS3D can be performed like in the 2D case in discrete or continuous coordinates and the result is again a *distance field*. Thus, when decompressing we can perform a *discrete extraction* adapted to 3D and obtain an approach to the initial total mesh with the resolution that is needed. Alternatively, we can define a new center, size and number of voxels; and extract a new *distance field* that it completely different to the original. In order to calculate its distance values, the IFS3D *continue extraction* will be used. Therefore, thanks to the fractal decompression fractal a *distance field* can be decompressed from the IFS3D with the resolution (see Figure 2(c)) and in the spatial position of the space that is desired.

**From 3D image to polygonal mesh:** Transforming a fractal decompressed 3D image into a polygonal mesh is very simple. All that has to be done is to reconstruct the triangles using *marching cubes* and treat the decompressed image as a *distance field*. Each triangle will be obtained from the *marching cubes* algorithm and thanks to the distance values contained in the voxels. The vertices will be adjusted to the original mesh surface by linearly interpolating the distance values of adjacent voxels.

## 4. Results

In order to test the new method, an experimental implementation of the described algorithms has been made. It has been used C++ with OpenGL and GLSL. All the test were executed on a single PC with a single AMD Sempron 2200 to 1.5 GHz processor with 512 MB of RAM and a NVIDIA GeForce 6200. All the executables are compiled with the GNU C/C++ Compiler.

Table 1 provides compression results for 2D, 2.5D and 3D IFS fractals. The images to compress are a grayscale Lenna image of 256x256, a 512x512 sample DEM also in grayscale named Terrain, an rgb image of 424x640 named Eagle, a

| Model & params | Time | Psnr | Ratio |
|---|---|---|---|
| Lenna const range2 | 14.000s | 49.79db | 0.81:1 |
| Lenna const range1 | 64.891s | 60.77db | 0.20:1 |
| Lenna quad 5000 | 11.750s | 37.74db | 2.56:1 |
| Lenna quad 10000 | 22.454s | 44.75db | 1.28:1 |
| Terrain const range2 | 373.172s | 67.92db | 0.79:1 |
| Terrain const range1 | 1553.766s | 72.28db | 0.20:1 |
| Terrain quad 20000 | 283.578s | 49.63db | 2.49:1 |
| Terrain quad 40000 | 501.359s | 53.86db | 1.24:1 |
| Eagle const range2 | 257.765s | 50.15db | 2.10:1 |
| Eagle const range1 | 1155.125s | 62.07db | 0.53:1 |
| Eagle quad 40000 | 228.469s | 47.67db | 3.33:1 |
| Eagle quad 80000 | 476.062s | 53.79db | 2.10:1 |
| Bunny const range2 | 14.844s | 39.74db | 6.93:1 |
| Bunny const range1 | 71.500s | 61.15db | 1.12:1 |
| Bunny quad 5000 | 42.875s | 39.71db | 13.87:1 |
| Bunny quad 10000 | 62.719s | 39.74db | 10.72:1 |
| Dragon const range2 | 578.828s | 42.70db | 9.36:1 |
| Dragon const range1 | 2801.938s | 63.36db | 1.47:1 |
| Dragon quad 20000 | 2187.797s | 42.55db | 26.63:1 |
| Dragon quad 40000 | 2963.063s | 42.70db | 20.49:1 |

**Table 1:** *Compression results of different sample fractals with a set of parameters.*

64x64x64 *distance field* of Bunny and another one of the *Phlegmatic dragon* with a 128x128x128 size. The compression parameters of the samples have been chosen by the authors between the most characteristic ones. They correspond to a constant regular structure with ranges of size 2 and 1 (quasi-lossless), and a structure in *quadtree* form with fixed number of transformations according to the size of each image. The domains set is without a doubt the bottle neck of the algorithm and affects in such a way that the more great is its number the more time costs the compression but the more quality is obtained. In all cases a domains set of fixed jump of 4 will be used because it has a good relation between resulting quality and time used in the compression. It can be deduced that as long as the number of transformations is raising, the quality of the fractal raises also and its ratio of compression descends. Obtaining even ratios that surpass the initial size of the file to compress. However, this does not imply that they are due to reject because the new format contributes to the image with fractal qualities that can justify their increase of size. Also it is convenient to indicate that when compressing *distance fields* greater compressions usually occurs because their great zones with identical values. This zones are those that lay completely inside or completely outside the original mesh.

Once the fractals from the previous table are compressed, Table 2 shows the results of their total decompression time in different situations. The *quadtree*-based fractal with more transformations is used as sample for each model. All the decompression tests have been made iterating the IFS until an *mean absolute error* ($E_{MAE}$) of less than 0.002 is reached between an iteration and the next one. This, rarely needs more than 10 iterations of the IFS. The used methods have been both the discreet method and the real coordinates one. The chosen zooms have been 50%, 100% and 200% in reference to the fractal original size. Regardless of the results, it is deduced in the first place that the decompression in real coordinates presents a computational weight that is greater than the discreet one. However, in the *distance fields* everything opposite happens because of the big internal and external zones of the mesh. With the continuous method these zones are instantly discarded whereas in the discreet method transformations are executed with them in all the iterations of the IFS.

| Model & params | Zoom | Discrete | Real |
|---|---|---|---|
| Lenna quad 10000 | 50% | 0.078s | 0.062s |
| Lenna quad 10000 | 100% | 0.125s | 0.250s |
| Lenna quad 10000 | 200% | 0.484s | 0.969s |
| Terrain quad 40000 | 50% | 0.218s | 0.344s |
| Terrain quad 40000 | 100% | 0.562s | 1.313s |
| Terrain quad 40000 | 200% | 2.078s | 4.945s |
| Eagle quad 80000 | 50% | 0.656s | 0.687s |
| Eagle quad 80000 | 100% | 2.047s | 2.530s |
| Eagle quad 80000 | 200% | 8.343s | 9.794s |
| Bunny quad 10000 | 50% | 0.047s | 0.016s |
| Bunny quad 10000 | 100% | 0.343s | 0.125s |
| Bunny quad 10000 | 200% | 2.766s | 0.984s |
| Dragon quad 40000 | 50% | 0.343s | 0.125s |
| Dragon quad 40000 | 100% | 2.219s | 0.922s |
| Dragon quad 40000 | 200% | 17.281s | 7.125s |

**Table 2:** *Decompression times of a set of fractals with discrete and continuous methods and zooms of 50%, 100% and 150%.*

It is very complicated to show the results that the new method has in the representation of meshes. Even more when it is wanted to make use of the characteristics of ROI, infinite resolution, progressive decompression or selective decompression in real time environments. We have implemented them all and can affirm that besides to work, they endow with flexibility the method to adjust it to any necessity. On the other hand, the fractal extraction is not as fast as some of the present multiresolution methods are. Nevertheless, the changes can get to be more than substantial in the appearance (see Figure 2).

Performance tests of fractal decompression shader have also been made. Nowadays, the GPU has many implementation and execution limitations. Even so, it is possible to be announced that the shader decompression works in real-time at full screen (1024x768) in restricted environments. This is if we limit the fractal to a 256x256 grayscale one, with con-

stant structure and with a fixed number of iterations of 10 or less. For instance, the fractal of the Figure 4.

## 5. Conclusions

This work presents a new multiresolution model entirely based on the fractal image compression techniques. A polygonal mesh is codified (compressed) as an IFS over a continuous domain. The model is progressively decodified (decompressed) by iterating the IFS. The codified model can be decodified at whatever level of detail and size, even levels of detail of higher resolution than those in the original model can be obtained. The codification is a lossy process, so the size of the codified model can be adjusted. Finally, lossless compression is always possible but this way the size of the codified model is higher that the original one.

Future research direction include improving the continuous recovery to take into account the subsampling with neighbours that the IFS needs. Another planned improvement is to directly obtain the fractal representation without using any *distance field*. We also plan to investigate paging the geometry and the textures of the model for very dense polygonal meshes, and the model streaming as well. As further investigation, we plan to add the time dimension to the codification process, in such a way that an animated mesh could be represented as a fourth-dimensional fractal. Finally, to compare the performance of our model with the performance of models with similar characteristics is a planned work as well.

## 6. Acknowledgements

We would like to thank J. Gumbau for his contribution to the fractal GLSL decompression shader. This work was supported by grant P1 1B2005-17 of Fundació Caixa Castelló-Bancaixa.

## References

[Bar93] BARNSLEY M. F.: Fractal modelling of real world images. In *Fractals Everywhere*, 2nd ed. Academic Pres, New York, NY, USA, 1993, pp. 219–239.

[Bar02] BARNSLEY M. F.: Iterated function systems for lossless data. *The IMA Volumes in Mathematics and its Aplications 132* (2002), 33–64.

[DP02] DECORO C., PAJAROLA R.: Xfastmesh: fast view-dependent meshing from external memory. In *VIS '02: Proceedings of the conference on Visualization '02* (2002), IEEE Computer Society, pp. 363–370.

[DWS*97] DUCHAINEAU M., WOLINSKY M., SIGETI D. E., MILLER M. C., ALDRICH C., MINEEV-WEINSTEIN M. B.: Roaming terrain: real-time optimally adapting meshes. In *VIS '97: Proceedings of the 8th conference on Visualization '97* (Los Alamitos, CA, USA, 1997), IEEE Computer Society Press, pp. 81–88.

[Ebe99] EBERLY D.: Distance between point and triangle in 3d. http://www.geometrictools.com.

[EMWVB01] ERIKSON C., MANOCHA D., WILLIAM V. BAXTER I.: Hlods for faster display of large static and dynamic environments. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), ACM Press, pp. 111–120.

[ESAV99] EL-SANA J., AZANLI E., VARSHNEY A.: Skip strips: maintaining triangle strips for view-dependent rendering. In *VIS '99: Proceedings of the conference on Visualization '99* (1999), IEEE Computer Society Press, pp. 131–138.

[FBMK95] FISHER Y., BARAHAV Z., MALAH D., KARNIN E.: *Fractal Image Compression: Theory and Application*. Springer Verlag, 1995.

[Fis92] FISHER Y.: *Fractal Image Compression*. Tech. Rep. 12, Department of Mathematics, Technion Israel Institute of Technology, 1992. SIGGRAPH '92 COURSE NOTES.

[FMP98] FLORIANI L. D., MAGILLO P., PUPPO E.: Efficient implementation of multi-triangulations. In *VIS '98: Proceedings of the conference on Visualization '98* (1998), IEEE Computer Society Press, pp. 43–50.

[FPM96] FLORIANI L. D., PUPPO E., MAGILLO P.: A formal approach to multiresolution modeling. In *Theory and Practice of Geometric Modeling* (1996), pp. 302–323.

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 209–216.

[Gum05] GUMHOLD S.: Truly selective polygonal mesh hierarchies with error control. *Comput. Aided Geom. Des. 22*, 5 (2005), 424–443.

[HDJ05] HWA L. M., DUCHAINEAU M. A., JOY K. I.: Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 355–368.

[Hop97] HOPPE H.: View-dependent refinement of progressive meshes. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 189–198.

[Hut81] HUTCHINSON J. E.: Fractals and self-similarity. *Indiana Univ. Math. J. 30* (1981), 713–747.

[Jac90] JACQUIN A. E.: Fractal image coding based on a theory of iterated contractive image transformations. *Proceedings of the SPIE - The International Society for Optical Engineering 1360* (Oct 1990), 227–239.

[JKK04] JUNHO KIM S. L., KOBBELT L.: View-

dependent streaming of progressive meshes. In *Proceedings of the shape modeling international 2004* (2004), pp. 209–220.

[Jon95] JONES M. W.: Voxelisation of polygonal meshes. In *Proceedings of the 13th annual conference of Eurographics (UK Chapter)* (March 1995), pp. 160–171.

[KB96] KAISER P. K., BOYNTON R. M.: *Human Color Vision*. Optical Society of America, May 1996.

[KL01] KIM J., LEE S.: Truly selective refinement of progressive meshes. In *GRIN'01: No description on Graphics interface 2001* (2001), Canadian Information Processing Society, pp. 101–110.

[KL03] KIM J., LEE S.: Transitive mesh space of a progressive mesh. *IEEE Transactions on Visualization and Computer Graphics 9*, 4 (2003), 463–480.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, July 1987), vol. 21, ACM Press, pp. 163–169.

[LE97] LUEBKE D., ERIKSON C.: View-dependent simplification of arbitrary polygonal environments. *Computer Graphics 31*, Annual Conference Series (1997), 199–208.

[Paj01] PAJAROLA R.: Fastmesh: Efficient view-dependent meshing. In *PG '01: Proceedings of the 9th Pacific Conference on Computer Graphics and Applications* (2001), IEEE Computer Society, p. 22.

[PD04] PAJAROLA R., DECORO C.: Efficient implementation of real-time view-dependent multiresolution meshing. *IEEE Transactions on Visualization and Computer Graphics 10*, 3 (2004), 353–368.

[PFMC04] PEREZ M., FERNÁNDEZ M., MORILLO P., COMA I.: Locally constrained synthetic lods generation for natural terrain meshes. *Future Generation Comp. Syst. 20*, 8 (2004), 1375–1387.

[Pri00] PRINCE C.: Progressive meshes for large models of arbitrary topology, 2000.

[XV96] XIA J. C., VARSHNEY A.: Dynamic view-dependent simplification for polygonal models. In *VIS '96: Proceedings of the 7th conference on Visualization '96* (Los Alamitos, CA, USA, 1996), IEEE Computer Society Press, pp. 327–ff.

[ZC06] ZHENG Z., CHAN T. K. Y.: Traversal on dag-based multiresolution mesh hierarchy. In *CGIV '06: Proceedings of the International Conference on Computer Graphics, Imaging and Visualisation* (2006), IEEE Computer Society, pp. 302–309.

```
uniform sampler2D texDOMPOS, texSO,
                  texORI, texCOEF;
uniform float seed,transx,invtransx;
uniform int iterations;

void main(void)
{
  int deep;
  float factor0, factolast;
  float factoracum=0.0;
  float a, b, c, d, e, f;
  vec2 auxv;

  vec2 posaux=gl_TexCoord[0].st;

  // first iteration
  auxv=texture2D(texSO,posaux).xy*2.0-1.0;
  factorultimo=auxv.y;
  factor0=auxv.x;

  // next iterations
  for (deep=1;deep<iterations;deep++)
  {
    f=texture2D(texORI,posaux).x;

    auxv=vec2(0.0625,f);
    a = texture2D(texCOEF,auxv).x*4.0-2.0;
    auxv=vec2(0.1875,f);
    b = texture2D(texCOEF,auxv).x*4.0-2.0;
    auxv=vec2(0.3125,f);
    c = texture2D(texCOEF,auxv).x*4.0-2.0;
    auxv=vec2(0.4375,f);
    d = texture2D(texCOEF,auxv).x*4.0-2.0;
    auxv=vec2(0.5625,f);
    e = texture2D(texCOEF,auxv).x;
    auxv=vec2(0.6875,f);
    f = texture2D(texCOEF,auxv).x;

    auxv=(vec2(e,f)+floor(posaux*transx))
         *invtransx;
    auxv=posaux-auxv;
    posaux=texture2D(texDOMPOS,posaux).xy;
    posaux+=vec2(a*auxv.x+b*auxv.y,
                 c*auxv.x+d*auxv.y);
    auxv=texture2D(texSO,posaux).xy*2.0
         -1.0;

    factoracum+=factorultimo;
    factorlast=factor0*auxv.y;
    factor0*=auxv.x;
  }

  gl_FragColor=vec4(seed*factor0+factoracum
              +factorlast);
}
```

**Figure 6:** *GLSL source code of the fractal pixel shader.*