Simulation and Rendering of *Opus Palladium* 3D Mosaics

Vladimir Alves dos Passos and Marcelo Walter
Centro de Informática - UFPE, Brazil

Abstract

Mosaic sculptures are a form of art where the sculpture is made from a collection of individual elements called tiles which are distributed over the surface of a given 3D shape. To add expressiveness, artists distribute the tiles following a high-level design over the shape, in order to emphasize some features. In the method known by mosaicists as Opus Palladium*, or simply "crazy paving", tiles with irregular shapes are used, improving the expressivity of the final result. In this work, we present a method to simulate this kind of 3D mosaic by representing tiles as Voronoi polygons computed from a distribution of points on the surface of the 3D object. Previous work on this topic have used only square-shaped tiles. Special mosaic-like effects are obtained with the help from texture maps, which control the high-level design of the tiles.*

1 Introduction

Traditional art forms are a source of inspiration for much computer graphics work. A lot of the work in two-dimensional expressive rendering, for instance, use as inspiration art forms such as oil painting [17, 8], watercolor painting [6, 18], pen-and-ink [24], and others [5, 16]. Less common are three-dimensional art forms, such as sculptures. Our work in this paper targets a particular type of sculptures – mosaic sculptures – where the artist covers a given 3D shape with a collection of small pieces called tiles. Tiles are usually made of ceramics or glass and can be used in regular or arbitrary shapes. Previous work on this topic have successfully addressed 3D or surface mosaics, as they are sometimes called, using tiles of the same shape and size [15] and, more recently, of the same shape but variable sizes [7].

Here we present a solution for the case where the tiles have an arbitrary shape. Among mosaicists [14] this type of mosaic is called *Opus Palladium*, or "crazy paving", a reference to the artistic freedom that comes from using arbitrarily-shaped tiles. In Figure 1 an example of this mosaic is given.

Figure 1. Example of a real opus palladium mosaic (detail from the Lizard sculpture in Park Guell, Barcelona).

In Figure 2 we show a result from our simulation on an eagle model. The placement, size and distribution of tiles were computed automatically. Parameters used in all simulations are given in Table 1.

2 Previous Work

In Computer Graphics, 2D mosaics have been explored since the early 1990s, when Haeberli [12] used Voronoi polygons to draw beautiful mosaic-like effects from images. From this seminal work a lot of progress has been made and S. Battiato and colleagues summarize the recent contributions in [2, 1]. According to this survey, there are three general types of 2D mosaics: ancient, photomosaics, and puzzle image mosaics. We are interested only in the first type, and our review below is restricted to this type of mosaics. In this type, the digital mosaics use as a source of inspiration actual mosaics. The mosaics themselves are composed by grouping together a collection of small pieces called tiles.

After the pioneering work by Haeberli, it was not until 2001 that mosaics attracted attention again in graphics research. Hausner [13] presented beautiful mosaics computed from a given image. The tiles are mostly square but rectangular tiles were also used. Tile positioning was obtained with an iterative process which moves the points representing the tiles towards a centroidal Voronoi diagram.

In 2003 Elber and colleagues [9] introduced a mosaicing technique where the tiles are arranged in rows along free-form feature curves computed for the image, much like level curves that spread outwards from a source. In the work

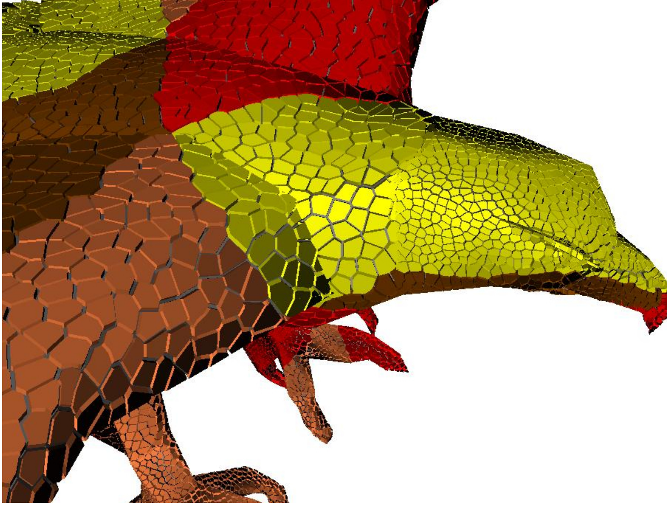


Figure 2. Example of result from our simulations. The size and placement of tiles automatically computed with our solution.

by Di Blasi and colleagues, titled Artificial Mosaics [4], the authors used inspiration from the work of mosaicists to drive their solutions with impressive-looking results. The work called *RenderBots* [19] uses a multi-agent approach for rendering mosaics. A number of RenderBots are distributed in the image. These bots are configured according to the desired visual effect. The bots can be configured in their physical behaviour to account, for instance, for different painting styles, among them mosaic-like. More recent work express the problem of artificial mosaic generation as a gradient vector flow computation [3].

Comparatively, there has been very little work addressing 3D or surface mosaics. The work presented in [15] posed the problem of tile placement on a surface as a global optimization problem. Their solution used square tiles of same size. In [7] we presented a solution for 3D mosaics where the tiles are still square, but the individual sizes are adjusted according to a local metric based on geometric properties of the surface receiving the tiles, such as approximated curvature. Our work in this paper shares many similarities with this last work, the main difference is that we compute tiles of many shapes, instead of only square-shaped ones.

In the next section we explain how we implemented our solution.

3 Method

Our main goal is to achieve variation in the shape of tiles. As a second goal, the distribution of tiles should follow a high-level design as seen in many of these types of mosaics, and illustrated in Figure 1. Our solution computes a distribution of tiles of variable size on the surface, subject to constraints imposed by the geometry, and also constraints imposed by the design, computed with the help from texture maps. Our solution is an extension of the work presented in [7], and therefore we will quickly review this work and then we will discuss our solution for the two above mentioned goals.

3.1 Distribution of square tiles of variable sizes

The solution presented in [7] computes square tiles with variable sizes in two steps, as follows:

3.1.1 Random Tile Distribution on the Surface of a Polyhedral Model

Tiles are represented as points, and an initial distribution of points over the model's surface is computed using a modified version of the algorithm presented by Turk in [21]. In Turk's approach, only the relative areas A_i of each polygon i are used for point placement. In the algorithm presented by Passos and Walter [7], point placement takes also into account the relative capacity C_i of each polygon. This capacity is a function not only of the polygon's area, but also of its curvature. With this extension, the initial configuration of tiles naturally places less tiles over flat areas, and more tiles over curved areas. A similar idea in 2D was presented by Faustino and Figueiredo [10], where the size of tiles was adaptively computed as a function of how close the tiles were from main features of the image.

In our approach, the curvature on each vertex was approximated with the algorithm presented also by Turk in [22]. Finally, a function f maps radii of curvatures to size ts of tiles. f is a simple linear function with two thresholds, ts_{min} and ts_{max}. The threshold values are given in terms of an average tile size $\overline{ts}$, which is proposed by [15] as the average tile size if we were to cover the entire surface using an user-specified number N of tiles with the same size.

3.1.2 Point Relaxation on the Surface

The step described above places tiles randomly on the surface. In order to achieve an even distribution over the surface, we use a relaxation process. This process will spread the points such that they are at approximately the same distance from one another. The algorithm considers each point as an interacting particle that produces a force field around it. This field is repulsive, so that points will repel each other. The repulsive force F_{ij} between points i and j is given according to equation below:

$$F_{ij} = \begin{cases} k_f(1 - \frac{d}{r_i + r_j}) & d \leq r_i + r_j \\ 0 & elsewhere \end{cases}$$

where r_i and r_j are the values obtained from the mapping function f applied to the radii of curvature on each point location, d is the distance between points i and j, and k_f is a parameter that controls the overall time for the system to reach a steady state. We used $k_f = 0.04$ in our results.

Measuring these distances on a mesh is not trivial, and an approximation presented in [23] is used. Basically, all computations are made on the supporting plane of the point being considered. For artistic purposes, using an approximated planar distance between the points is not critical. With an increase in cost we could use more sophisticated solutions for computation of geodesics, as proposed in [20].

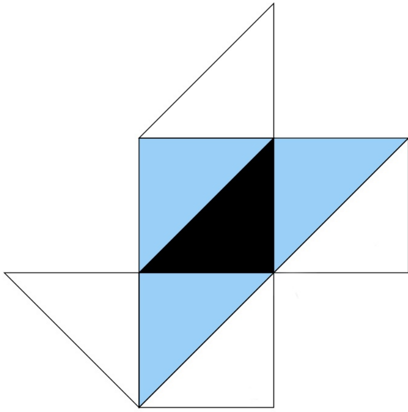


Figure 3. Section of a flattened cube. For the black triangle we show the primary (blue) and secondary (white) neighbors.

With this approximation, the only neighboring points considered are the ones located in either primary (share an edge), or secondary faces (share a vertex), with the supporting polygon. Figure 3 illustrates this idea. The black triangle is the reference triangle. All particles will be mapped to the supporting plane of this triangle. The blue triangles are the primary neighbors and the white triangles are the secondary neighbors. The number of primary and secondary neighbors is not fixed and depends on the topology of the mesh.

The distances are computed on the supporting plane and therefore the algorithm needs to map the primary and secondary neighbors to this plane. For each pair of faces R and S that share an edge, we precompute two matrices $M_{R \to S}$ and $M_{S \to R}$ that operate the rotation from one plane into another. For the primary neighbors we use these matrices. For secondary neighbors, we use a sequence of rotations around edges which are shared by the supporting face and the secondary neighbor being projected. In order to update the position of points due to the relaxation forces, points travel freely and can eventually move to another face. When a cell changes face we find which edge the cell crossed and using the precomputed rotation matrices we bring the point's position onto the plane of the new face. This process is repeated until the point rests on some face, as illustrated in Figure 4.

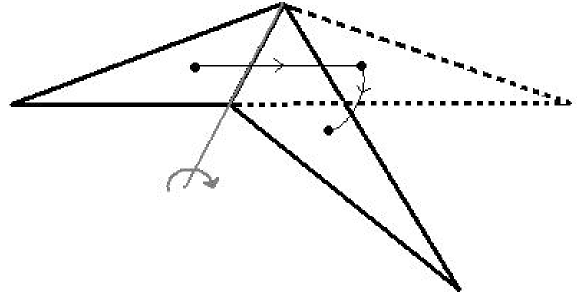


Figure 4. A point pushed away from its original face is rotated around the crossed edge until it falls over the neighboring face.

3.2 Rendering Variable-Shaped Tiles Using Voronoi Diagrams

In this work, we propose to simulate the crazy paving style using Voronoi polygons to represent individual tiles. Voronoi polygons have enough shape variation and are a good candidate for tiles with variable-shape. At this stage of the algorithm, for each polygon on the mesh, we have a collection of evenly spaced points that represent the tiles. As in the relaxation step explained above, we use an approximation for the computation of the Voronoi diagram presented in [23]. We compute the Voronoi diagram of this collection of points on the plane of the supporting polygon, and map all neighboring faces to this same plane. Again, since our application is mainly artistic, it makes sense to use an approximation instead of the full cost of an exact computation.

An important visual component of mosaics is the space among tiles, filled with grout. Our Voronoi computation results in tiles with no space among them. We introduce a global scaling factor in order to simulate and control the amount of grout. We decrease globally the size of each tile according to a user-supplied amount of grout g, a number between 0 and 1. This reduction in size is computed with a simple scaling of the tile in its local frame of reference. We have used g typically as 10%. In order to give a more realistic feel, one could vary randomly or according to tile sizes or shapes (restricted to a range of values) the amount

of grout between the tiles. We will implement this as future work.

The size difference between the initial and the final shape of the tile is described as a collection of quadrilateral polygons, and should be included on the mesh representing the full mosaic sculpture, as shown in Figure 5. These grout polygons receive a flag identifying them as such, in order to render them differently from mosaic polygons.

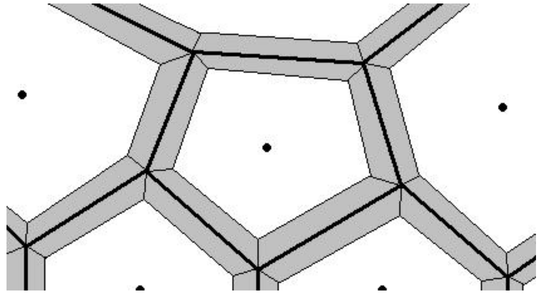


Figure 5. Grout generated after tile reduction.

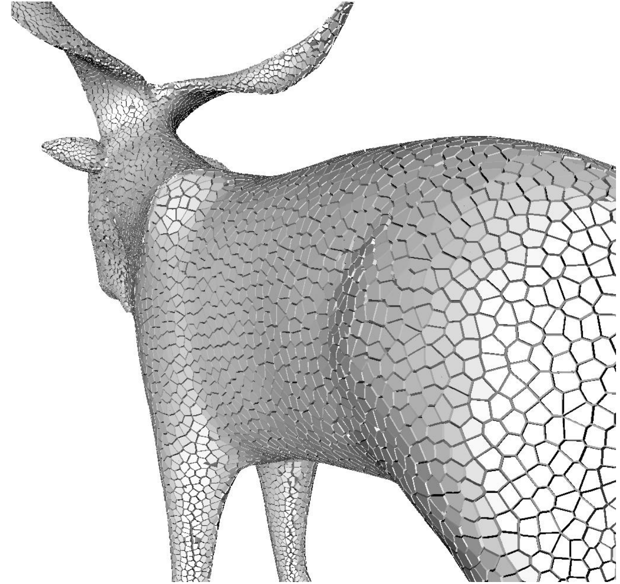


Figure 6. Mosaic generated according to crazy-paving style.

Once we have computed all grout polygons, we clip the entire set against the supporting polygon using the Sutherland clipping algorithm [11]. The resulting set is then projected back to 3D. Once we have all tiles and grout polygons, if needed, we are able to discard the original mesh. At the rendering step, polygons identified as grout are rendered using a gray color, and those identified as tiles will use the reference to the point that generated it. This reference contains texture information, which will determine the color of the tile, and also contains the normal N at that point on the original mesh. The normal vector is used for computing the height of tiles. Each tile's vertex will be displaced by some value in the direction of N. Figure 6 shows a result without texture information yet.

3.3 Control of the design

Most 3D mosaics exhibit a high-level design for tile positioning. As illustrated in Figure 1, tiles are positioned following bands of different colors. In order to implement this style, we use texture maps for specification of the design and implemented a small modification in the particle relaxation process: we force the tiles to align with the edges present in the texture map.

The problem of tile alignment for 2D mosaics was investigated by Di Blasi and Gallo [4], where a solution based on directional guidelines was presented. In their work the directional guidelines are the main features of the image and from these they compute three matrices: a distance transform matrix, a gradient and a level line matrix. These last two matrices are used to align the tiles.

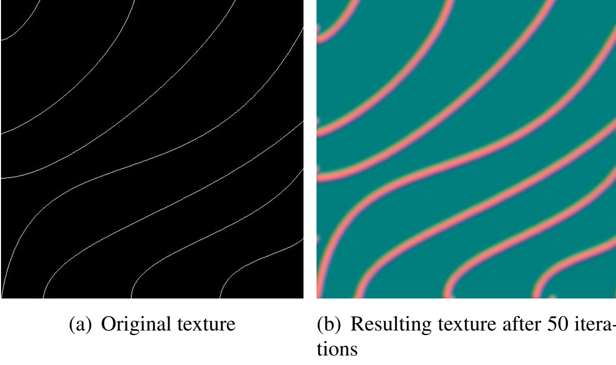
Our idea is to include in the force field artificial forces due to the edges present in the texture map. These forces should repel the particles from the locations that are mapped near the edges on the texture. Therefore, the closer the point is to the edge, stronger is the force. The information of this distance, between the point and the closest edge, is found in a second texture that is pre-computed according to the following process, illustrated in Figure 7.

First, we obtain a black/white representation of the edges from the image (Figure 7(a)). This might be automaticaly produced using any edge detection algorithm, or it could be hand-made by the user. Edges are assumed to be white, and background black. Then we iterate over this image m times, and for each pixel, its color will be diffused amongst its 4-neighbors – up, down, left and right. The diffusion rate d_r will determine the velocity of diffusion, and m will determine the range. This process is equivalent to applying a weighted mean filter on the image.

At the iteration k, pixel's color $P_k(i,j)$ at coordinates i and j is evaluated as:

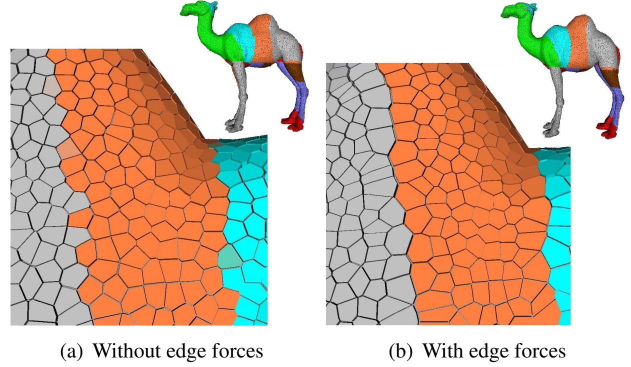
$$P_k(i,j) = \frac{d_r}{4}S + (1 - d_r)P_{k-1}(i,j)$$

where S is the sum of the 4-neighbor's color. The value is then saved into the red component of the new texture. The closer the pixel is from the edge, the higher is its red value. In the green and blue components of the image, we save the gradient information from the x and y directions. These values are evaluated as the differences between the two neighbors' red value on each direction, normalized to



(a) Original texture (b) Resulting texture after 50 iterations

Figure 7. Texture for controlling the design. $d_r = 0.7$ **and** $m = 50$



(a) Without edge forces (b) With edge forces

Figure 8. Comparison between mosaics without and with edge control.

fit in the range $[0, 255]$. The equations below compute the gradient in the blue $B(i, j)$ and green $G(i, j)$ components respectively, of the new texture:

$$B(i,j) = \frac{R(i+1,j) - R(i-1,j) + 255}{2}$$

$$G(i,j) = \frac{R(i,j+1) - R(i,j-1) + 255}{2}$$

Zero values of the gradient will be mapped to 177.5, rounded to 177. Negative values will be in the range $[0, 176]$ and positive values in the range $[178, 255]$. Figure 7(b) shows the resulting texture computed as explained above with $d_r = 0.7$ and $m = 50$.

Once we have the texture with the information about the gradient, we are able to use it in our particle relaxation method. For each particle, we query the blue and green components from the computed texture. Subtracting 177 from each, we obtain directly the force vector in the coordinate system of the texture. We then map this vector into the coordinate system of the base polygon and add it to the resulting repulsive force due to the neighboring particles. This force can be multiplied by a scaling factor w that better suits the particular number of tiles being used. Particles will be pushed away from the edges, and due to repulsive forces between them, they will be forced to align near edges.

After the Voronoi computation, as illustrated in Figure 8, the tiles are properly aligned with the edges from the texture, and the final configuration still follows the *opus palladium* mosaic style.

4 Results

In this section we illustrate a few results of our technique. In Table 1 we list the parameters used: number of tiles, mapping of tile sizes to curvatures, and amount of grout. Timings on a $2.26Ghz$ processor with $1.25GB$ RAM are around $10sec$ for 15000 tiles, mostly due to computation of the Voronoi cells.

Table 1. Parameters and their values for the results presented.

Figure	# of tiles	ts_{min}	ts_{max}	g
2	20000	1 $\overline{ts}$	4 $\overline{ts}$	0.1
6	17000	0.5 $\overline{ts}$	1.4 $\overline{ts}$	0.1
9	5000	1 $\overline{ts}$	1 $\overline{ts}$	0.1
10 / 11	15000	0.5 $\overline{ts}$	3 $\overline{ts}$	0.1
12	20000	0.5 $\overline{ts}$	1.5 $\overline{ts}$	0.15

Our renderings emphasize just the overall distribution of tiles and did not use any particular realistic rendering material model, except in Figure 11 where a wood-like material was used.

First we show a result on a mesh with the same overall curvature, a sphere. In Figure 9, we can see that the planar approximation of the Voronoi computation handles well the covering of the mesh. The Voronoi tiles spread over more than one polygon from the original mesh but nevertheless there are no visible discontinuities on the Voronoi cells. The bear mosaic sculpture in Figure 10 illustrates the variation in size of tiles on the 3D surface according to the approximated local curvature of the mesh. In this example we did not use any texture information, assigning random colors to the tiles.

For Figure 11 we saved the final mosaic sculpture as a mesh and loaded it in 3D Studio Max for the realistic rendering. The same distribution of tiles from Figure 10 was used, but with an uniform color for all tiles instead, a wood-like material property, and special lighting effects.

The final image in Figure 12 shows a dinosaur model with texture information defining the different parts of the

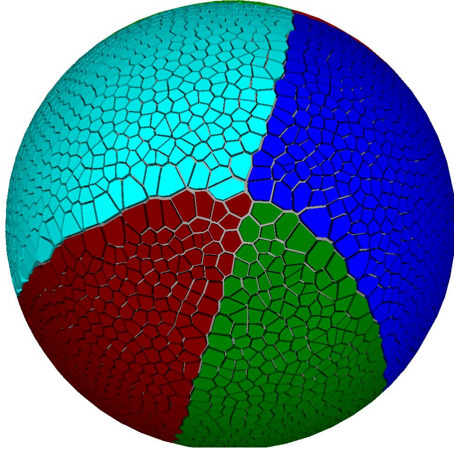


Figure 9. Example of our 3D mosaic on a sphere.

mosaic. Notice the slightly higher amount of grout used, 15% instead of 10%.

5 Conclusions

We presented a technique for building 3D textured mosaics with variable-shaped tiles. Previous work on this topic have only used squared-shaped tiles. The tiles are computed using Voronoi polygons to simulate the crazy paving style, and are positioned according to a high level control provided by texture maps. The tiles are also of variable size, adapted to the local geometry of the surface, and the amount of grout, or space among tiles, is precisely controlled. One drawback of our approximated Voronoi polygons is that, for some configurations of the original mesh, the primary and secondary neighbors do not take into account all possible neighbors. Therefore, some tiles are missed from the computation, causing discontinuities in the tiles. This can be avoided at the expense of including more faces in the search space.

For future work we are investigating hybrid mosaics, where there is a combination of square and variable-shaped tiles. Finally, we would like to render the tiles with material properties of real tiles, such as glass and ceramics.

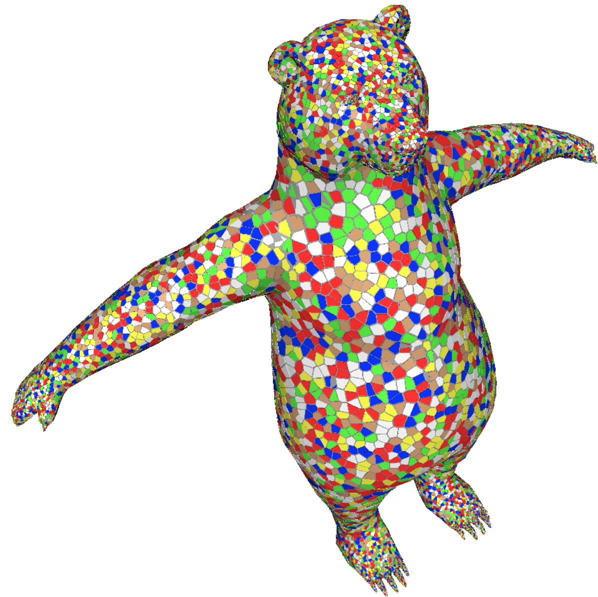


Figure 10. A 3D mosaic bear sculpture. Notice the variation in size of tiles on the body.

References

[1] S. Battiato, G. D. Blasi, G. Farinella, and G. Gallo. A survey of digital mosaic techniques. In *Proceedings of Eurographics Italian Chapter*, 2006.

[2] S. Battiato, G. D. Blasi, G. M. Farinella, and G. Gallo. Digital mosaic frameworks - an overview. *Computer Graphics Forum*, 26(4):794–812, 2007.

[3] S. Battiato et al. Artificial mosaics by gradient vector flow. In *Procedings of Eurographics 2008 - Short Papers*, 2008.

[4] G. D. Blasi and G. Gallo. Artificial mosaics. *The Visual Computer*, 21(6):373–383, 2005.

[5] T. W. Bleser, J. L. Sibert, and J. P. McGee. Charcoal sketching: Returning control to the artist. *ACM Transactions on Graphics*, 7(1):76–81, 1988.

[6] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, and D. H. Salesin. Computer-generated watercolor. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 421–430, Aug. 1997.

[7] V. Dos Passos and M. Walter. 3d mosaics with variable-sized tiles. *The Visual Computer*, 24(7-9):617–623, 2008.

[8] F. Drago and N. Chiba. Painting canvas synthesis. *The Visual Computer*, 20(5):314–328, 2004.

[9] G. Elber and G. Wolberg. Rendering traditional mosaics. *The Visual Computer*, 19(1):67–78, 2003.

[10] G. Faustino and L. de Figueiredo. Simple adaptive mosaic effects. In *Proceedings of SIBGRAPI 2005*, pages 315–322, Oct. 2005.

Figure 11. Wood bear sculpture. Same tiles as in Figure 10.

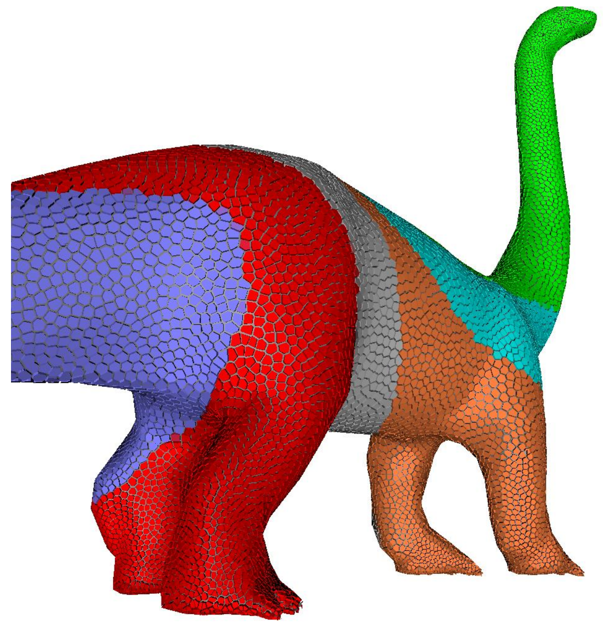


Figure 12. A dinosaur model.

[11] J. D. Foley et al. *Computer Graphics: Principles and Practice in C*. Addison-Wesley Professional, 1995.

[12] P. Haeberli. Paint by numbers: Abstract image representation. In *Proceedings of SIGGRAPH 1990*, pages 207–214, August 1990.

[13] A. Hausner. Simulating decorative mosaics. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 573–578, Aug. 2001.

[14] S. King. *Mosaic - Techniques and Traditions*. Sterling Publishing Co., 2003.

[15] Y.-K. Lai, S.-M. Hu, and R. R. Martin. Surface mosaics. *The Visual Computer*, 22(9-10):604–611, 2006.

[16] H. Lee, S. Kwon, and S. Lee. Real-time pencil rendering. In *NPAR*, pages 37–45, 2006.

[17] K. Lee et al. Three-dimensional oil painting reconstruction with stroke based rendering. *The Visual Computer*, 23(9-11):873–880, 2007.

[18] T. Luft and O. Deussen. Real-time watercolor illustrations of plants using a blurred depth test. In *NPAR*, pages 11–20, 2006.

[19] S. Schlechtweg et al. Renderbots - multi-agent systems for direct image generation. *Computer Graphics Forum*, 24(2):137–148, 2005.

[20] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.*, 24(3):553–560, 2005.

[21] G. Turk. Generating random points in triangles. In *Graphics Gems*, pages 24–28, 649–650. 1990.

[22] G. Turk. Re-tiling polygonal surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, pages 55–64, July 1992.

[23] M. Walter, A. Fournier, and D. Menevaux. Integrating shape and pattern in mammalian models. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 317–326, Aug. 2001.

[24] G. Winkenbach and D. H. Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 91–100, July 1994.