Hacettepe Journal of Mathematics and Statistics
Volume 37 (2) (2008), 107 – 114

INTRODUCING ISO-TILER 3D: A 3D TILING VISUALIZER

Haşmet Gürçay*

Received 01 : 07 : 2008 : Accepted 14 : 09 : 2008

Abstract

This paper introduces a Java applet called **Iso-Tiler 3D** that models whole tiling in 3D by using a single unit tile, and stores the model obtained in a 3D file.

Keywords: Tiling, Isohedral tiling, Tiling visualizer, JavaView, Periodic tiling.

2000 AMS Classification: Primary 52 C 22, 52 C 20; Secondary 68 D 18, 68 U 05.

1. Introduction

The history of tiles goes back to the prehistoric period. Coloured tiles can be seen in every civilization and culture as the most ancient decorative art. For example, Figure 1 represents an example of tiling in the church of Saint Nicholas, Myra (Turkey) [12]. Mathematically, a tiling is a geometric pattern made up of one or more shapes which fit together to completely cover an infinite plane region or surface without any gaps or overlapping.

Tiles are important tools for tiling the floors and walls in Computer Graphics. Several authors have explored the possibility of creating tilings in various forms by computer. C. Kaplan and D. H. Salesin [8] introduce and present a solution for the "Escherization" problem. They describe a representation for isohedral tilings that allows for highly interactive viewing and rendering. Later on, C. Kaplan and D. H. Salesin [10] showed how the original Escherization algorithm can be adapted to the dihedral case, producing tilings with two distinct shapes. In Kaplan [7] one procedure is described for constructing islamic star patterns based on placing radially-symmetric motifs in a formation dictated by a tiling of the plane.

*Hacettepe University, Department of Mathematics, 06800 Beytepe, Ankara, Turkey.
E-mail gurcay@hacettepe.edu.tr

Figure 1. Church of Saint Nicholas of ancient Myra (6th century AD)

Recently, the number of studies regarding 3D tiles have increased. For example, Cohen *et.al* [3] have focused on Wang tiles. Wang Tiles are squares in which each edge is assigned a color. A valid tiling requires all shared edges between tiles to have matching colors. They present a new stochastic algorithm to non-periodically tile the plane with a small set of Wang Tiles at runtime. Furthermore, they present new methods to fill the tiles with 2D texture, 2D Poisson distributions, or 3D geometry to efficiently create at runtime as much non-periodic texture as needed. Their use of Wang Tiles as containers for 3D geometry. Lu [11], extends the non-periodic tiling process of Wang Tiles to Wang Cubes and modifies it for multipurpose tiling. They automatically generate isotropic Wang Cubes consisting of 3D patterns or textures to simulate various illustrative effects.

Yen and Séquin [18] have developed an interactive program to design and manufacture "Escher Spheres" - sets of tiles that can be assembled into spherical balls in 3D. We also observe that various computer software systems for tiling have been developed. Some software such as JavaKali [2], Tess [15] and ScienceU - Triangle Tiler [13] that create the whole tiling by using a given unit tile have been produced. Tiling models that are obtained by this software are 2D graphical models. We have succeeded in creating tiling as a 3D graphical model and storing the model obtained in well known file types in 3D format. Thus, this model can easily be used by 3D artists and game programmers.

2. Mathematical Background

In this section we present some background on that part of tiling theory which is necessary for the implementations of our program Iso-Tiler 3D. The reader may obtain more detailed information from [1, 4, 5, 9, 14].

According to the definition of Kaplan [9], a *tiling* is a countable collection T of tiles $\{T_1, T_2, \ldots\}$, such that:

- Every tile is a closed topological disk.
- Every point in the plane is contained in at least one tile.
- The intersection of every two tiles is empty, a point, or a simple closed curve.

- The tiles are uniformly bounded; that is, there exist $u, U > 0$ such that every tile contains a closed ball of radius u and is contained in a closed ball of radius U.

When two tiles intersect in a curve, we may then refer to this well-defined curve as a *tiling edge*. Every tiling edge begins and ends at a *tiling vertex*, a place where three or more tiles meet. If every tile in a tiling is congruent to some shape T, we say that the tiling is *monohedral*, and that T is the *prototile* of the tiling. More generally, a *k-hedral* tiling is one in which every tile is congruent to one of k different prototiles. A tiling can be symmetric. A *periodic tiling* is a tiling of the Euclidean plane with periodic symmetry. That is, there exist two linearly independent directions of translational symmetry. In addition to a fundamental region, every periodic tiling has a *translational unit*, which is a fundamental region of the translational subgroup of the tiling's symmetry group [10].

For two congruent tiles T_1 and T_2 in a tiling, there will be some rigid motion of the plane that carries one onto the other (there may in fact be several). A somewhat special case occurs when the rigid motion is also a symmetry of the tiling. In this case, when T_1 and T_2 are brought into correspondence, the rest of the tiling will map onto itself as well. We then say that the two tiles are *transitively equivalent*. Transitive equivalence is an equivalence relation that partitions the tiles into *transitivity classes*.

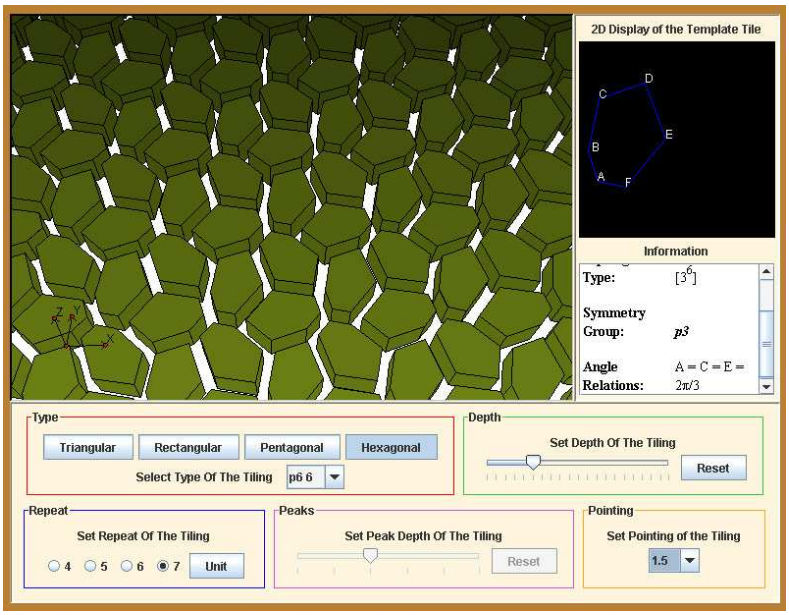
When a tiling has only one transitivity class, we call the tiling *isohedral*. More generally, a k-isohedral tiling has k transitivity classes. An isohedral tiling is one in which a single prototile can cover the entire plane through repeated application of rigid motions from the tiling's symmetry group. In an isohedral tiling, there is effectively no way to tell any tile from any other. Hence in an isohedral tiling there is essentially just one type of tile. An isohedral tiling is necessarily periodic. We now assume that we have a periodic tiling and we wish to establish its automorphism group. It is a surprising fact that there are only 17 different possible automorphism groups arising from periodic tilings, the so-called (plane) crystallographic groups or wallpaper groups [10]. The reader can see the 17 different wallpapers group on the web pages in Wikipedia [16].

3. Iso-Tiler 3D

Iso-Tiler 3D, written in Java, is software that uses the geometric software library of JavaView [6]. Therefore, Iso-Tiler 3D can use prevailing 3D geometric viewer features of JavaView. JavaView is a numerical software library with a 3D geometry viewer written in Java which was developed at Berlin Technical University. It allows the addition of interactive 3D geometries to any HTML document, and the presentation of numerical experiments online. Our Iso-Tiler 3D classes are inherited from JavaView classes. Iso-Tiler 3D is a Java applet that can model isohedral tiles that are deltahedral, tetrahedral, octahedral, and hexahedral polygons as 3D depending on their symmetry groups. In this 3D viewer (see Figure 2) program the user can be in a mutual interaction with the model (scale, translate, rotate, triangularize). It is mostly designed for artists and game programmers. The user can model an isohedral tile in 3D shape and store it in a file of *vrml, obj, mpl, byu, eps* format. It is then possible to operate with these formats using other programs.

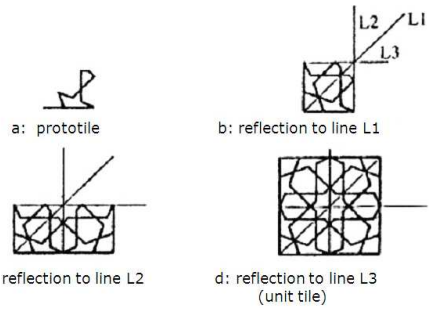
3.1. Implementation. The Iso-Tiler 3D program uses the symmetry features of tiles. If we know which element is included in the symmetry group of 17 different tiles, we can form all the periodic tiles and consequently the tiling. Iso-Tiler 3D can form the remaining tiling by using only one unit tile thanks to the required transformations. The tiling pieces are formed from polygons as JavaView library uses the corner points in forming the figures.

Figure 2. The interface of the Iso-Tiler 3D program



In our algorithm, prototile is actually a tiling piece. This tiling piece is transformed to a unit tile following a process of rotation, reflection and glide reflection in conformance with the periodical group of the tile. Then, the required shifts are performed to form a tile depending on the pattern of the periodical group and the whole tile is formed by the required transformations. See Figure 3.

Figure 3. Production of an sample unit tile

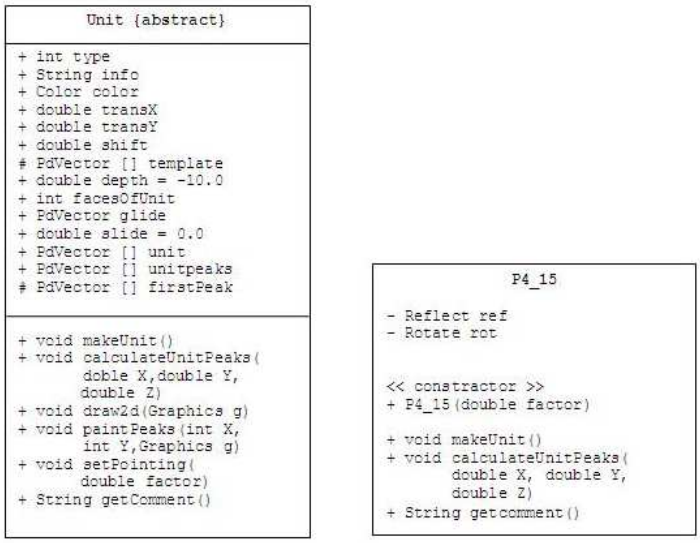


The coordinates of the vertices of the model to be formed on the viewer panel and data array that has the information on the surfaces of each vertex have been used on the basis of the algorithms used in Iso-Tiler 3D. JavaView basically needs two different arrays to show a three dimentional model. The first one is the array that includes the 3D coordinates of the vertices of the required figure. The other array store the surface information. Thanks to those two arrays, JavaView can form the required figure. The main aim here is to form those two arrays that are needed by JavaView and to change them, if required. JavaView will complete the mission.

It is clear that there are some variables to be defined to form a tile. The *tiler3d.parameter* program package has been formed for those variables. This package includes

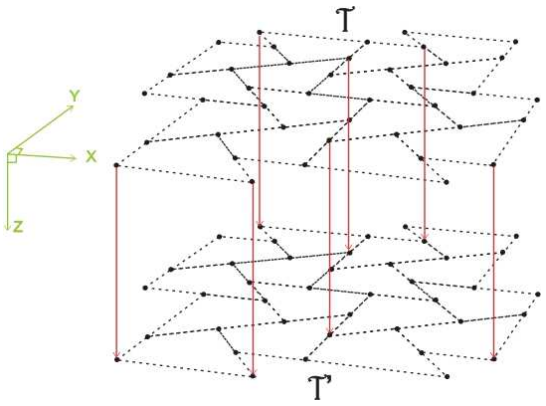
parameter classes that keep the features of each tile within its variables and operate on them using some special methods. Each parameter class has been formed by developing an abstract class called *Unit* which is included in the *tiler3d.parameter* package. This class contains common features and variables for all parameter classes. Each parameter class has been defined separately. Figure 4 presents the p4_15 parameter class.

Figure 4. The UML diagram of the *Unit* abstract class and the p4_15 parameter class



The *tiler3d.transform* package includes classes such as *Rotate2, Rotate3, Rotate4, Rotate6* and *Reflect* in order to determine the coordinates of the vertices of the unit tile. The first four of those classes are used to fulfill their $\pi, 2\pi/3, \pi/2$ and $\pi/3$ radiant rotations, respectively. As any rotation except those rotations has not been defined for tiles, only those four classes fulfill the rotation. The other class is the *Reflect* class, which is to be used for reflection and glide reflection processes. Those transformation classes use the *PuReflect* and *PdMatrix* classes that are included in the *jv.vecmath* package.

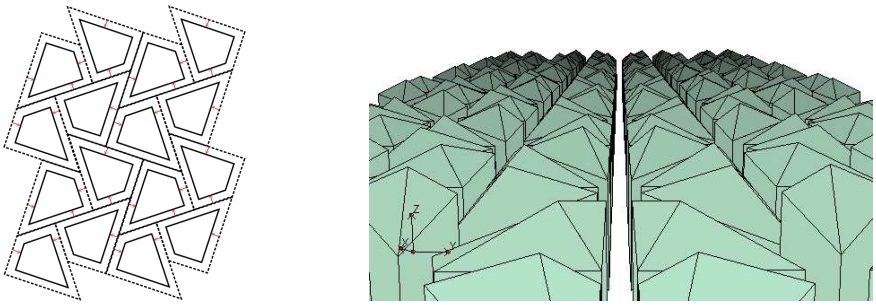
Figure 5. Obtaining a 3D model by using the *depth* factor



Following the formation of vertices in unit tile, the vertex points should be shifted through the pattern structure that fits the tile. In this way, all the vertex points of the tile in 2D format will be formed. The required size can be determined by repeating the shifting process. The shifting process is carried out by an object formed from the *engine.Translated* class within the *tiler3d.engine* package. This class is the basis class for the following classes. That is, the *Faced* class is written by developing the *Extruded* class, which in turn is written by developing *Translated* class.

The vertex points of the tile to be modeled is kept within an array called *translated*. This array also stores the required information for *Extruded*, which is an upper class of the *Translated* class. All the operations carried out up to the *Extruded* class are at the $z = 0$ level. For a 3D figure, an array that has twice as many elements as the array called *translated* is created. The contents of the *translated* array is copied to the first half of this array. In the second half, the value of the depth variable is subtracted from the z values of the terms and the estimation is copied. Thus, half of the points in the extruded array are at the $z = 0$ level, the other half is at $z =$ depth level. Following this process, all the vertices of the 3D tiling are obtained (see Figure 6).

Figure 6. Determining the dead space for P4-15 tiles



The program leaves an dead space between the tiles in order to increase the artistic effect of the tiles and to make it easier for the user to use the program. This is achieved by zooming out the tile pieces to some extent. For this purpose, a class called *tiler3d.engine.Shrinker* has been written. The dead space is determined by the *factor* value that is given as an argument to the configuration of the parameter class (see Figure 6).

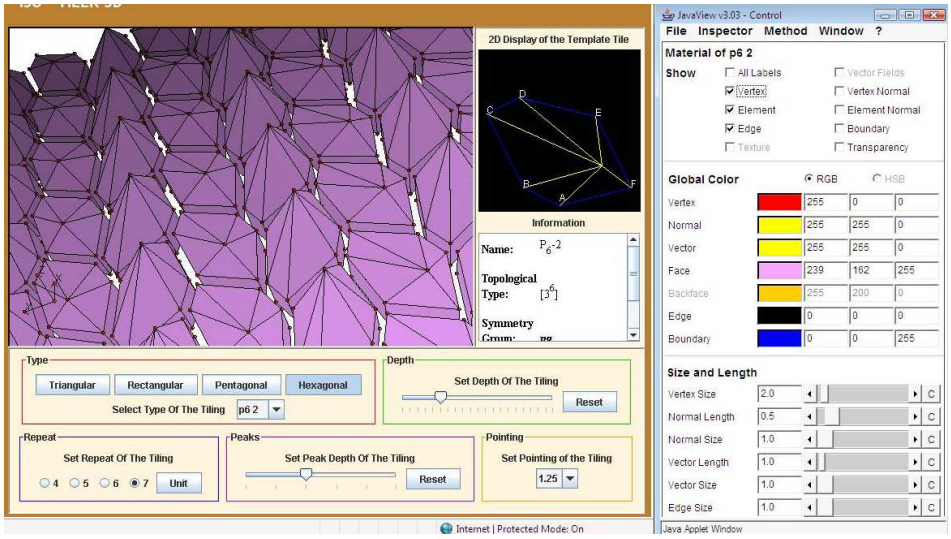
The *Project* and *Show* classes that enable the tiling to be seen on the screen have been included in the *tiler3d.display* package. The *Project* class is derived from the *jv.project.PjProject* class. The *PjProject* class is the main project class within Javaview. The aim of the project class is to control the geometry that is to be formed.

The *tiler3d.display.Show* class is the main conveyor derived from the *javax.swing.JApplet* class. As the *Show* class is a *JApplet* class, the *init()*, *start()*, *stop()* and *destroy()* methods respectively have been overwritten on the same methods in the *JApplet* class and have been redefined.

There are 12 classes within the *tiler3d.panels* package. This package has been created for panel classes that includes the components with which the user interacts. The *TypePanel* which consists of components that change the tiling, the *RepeatPanel* that fixes the repeat number of the tile, the *DepthPanel* that sets the depth of the tile, the *PeakPanel* that fixes the height of the pyramid and the *PointingPanel* that sets the grouting of the tiles have been derived from the *Jpanel* class. In order to reflect the changes made on the

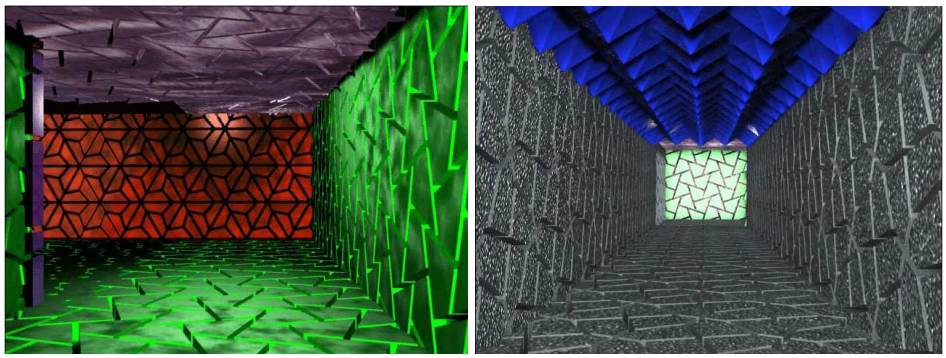
panels to the tiles, each operation starts a task. For those tasks, the *ChangeType*, *ChangeRepeat*, *ChangeDepth* and *ChangePeak* classes that are derived from *java.lang.Thread* have been written. All of those panels are installed into another panel called the *SettingPanel* that is also derived from the *javax.swing.JPanel* class. There is also another class called *InfoPanel*. This class contains a panel for a 2D image of the tile and another panel for an explanation window. All the changes made on the panels mentioned are captured by inner classes defined in *SettingPanel* and the required changes are made by calling the methods within the *Project* class.

Figure 7. The Iso-Tiler 3D interface with the JavaView Control Panel



3.2. Application. Iso-Tiler 3D not only enables the user see the object in 3D, it also transforms the tiles shown into 3D shapes and allows other 3D modeling programs use those images. The video images of a sample tiling file created by Iso-Tiler 3D and animated by the 3D Studio Max 7 modeling program are presented in Figure 8. The AVI file is available at the following address:
http://www.yunus.hacettepe.edu.tr/˜gurcay/HASMET/index_files/Page430.htm

Figure 8. Application



References

[1] Abas, S. J. and Salman, A. *Geometric and Group-Theoretic Methods for Computer Graphic Studies of Islamic Symmetric Patterns*, Computer Graphics Forum **11** (1), 43–53, 1992.

[2] Amenta, N. and Phillips M. *Java Kali*, Available online at: http://www.geom.uiuc.edu/java/Kali/ (accessed February 2007).

[3] Cohen, M. F., Shade J., Hiller S. and Deussen O. *Wang Tiles for image and texture generation*, International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH (San Diego, California, 2003), 287–294.

[4] Grünbaum, B. and Shephard, G. C. *Tilings And Patterns* (W. H. Freeman, New York, 1986)

[5] Jablan, S. V., *Symmetry And Ornament*, Available online at: http://www.emis.de /monographs/jablan/index.html (accessed February 2007).

[6] JAVAVIEW. Available online at: http://www.javaview.de/ (accessed February 2007).

[7] Kaplan, C. S. *Computer generated Islamic star patterns* In: *Bridges, Mathematical Connections in Art, Music, and Science*, Proc. of the 2000 Bridges Conference, ed. Reza Sarhangi (Southwestern College, Kansas, 2000).

[8] Kaplan, C. S. and Salesin, D. H. *Escherization*, SIGRAPH 2000, The 27th International Conference on Computer Graphics and Interactive Techniques (New Orleans, USA, (200)), 25–27.

[9] Kaplan, C. S. *Computer Graphics and Geometric Ornamental Design* (PhD Thesis, Uniersity of Washington, 2002).

[10] Kaplan, C. S. and Salesin D. H. *Dihedral Escherization*, Canadian Human-Computer Communications Society, in: the Proceedings of Graphics Interface (London, Ontario, 2004)

[11] Lu, A., Ebert, D. S., Qiao, W., Kraus, M. and Mora B. *Volume Illustration Using Wang Cubes*, ACM Transactions on Graphics **26** (2) Article 11 (June 2007).

[12] Lycian Turkey - Discover the Beauty of Ancient Lycia, http://www.lycianturkey.com /lycian_sites/myra.htm (accessed June 2007)

[13] ScienceU, *Tilings And Tesselations*, Available online at: http://www.scienceu.com/ (accessed February 2007).

[14] Sugimoto, T. and Ogawa, T. *Tiling Problem of Convex Pentagon*, FORMA **15** (1), 75–79, 2000.

[15] Tess, P edagogugery Software Inc., Available online at: http://www.peda.com/tess/ (accessed February 2007).

[16] Wallpaper Group. Available online at:http://en.wikipedia.org/wiki/Wallpapergroup (accessed December 2007).

[17] Xah, L. *The Discontinuous Groups of Rotation and Translation in the Plane*, Available online at: http://www.xahlee.org/Wallpaper_dir/c0_WallPaper.html (accessed February 2007).

[18] Yen, J. and Sequin, C. *Escher Sphere Construction Kit*, Symposium on Interactive 3D Graphics, 95–98, 2001.