Vis Comput (2011) 27: 555–563
DOI 10.1007/s00371-011-0589-4

ORIGINAL ARTICLE

Modeling with blocks

Luc Leblanc · Jocelyn Houle · Pierre Poulin

Published online: 22 April 2011
© Springer-Verlag 2011

Abstract This paper presents a simple and general modeling primitive, called a *block*, based on a generalized cuboid shape. Blocks are laid out and connected together to constitute the base shape of complex objects, from which is extracted a control mesh that can contain both smooth and sharp edges. The volumetric nature of the blocks allows for easy topology specification, as well as CSG operations between blocks. The surface parameterization inherited from the block faces provides support for texturing and displacement functions to apply surface details. A variety of examples illustrate the generality of our blocks in both interactive and procedural modeling contexts.

Keywords Subdivision surface · Surface parameterization · Polycube map · z-brush · Displacement map · Geometry image · CSG

1 Introduction

Modeling objects must satisfy different requirements, depending on the objects' surface properties, topology constraints, and on the modeling process itself. Through the years, several surface representations have been introduced, including polygonal meshes, subdivision meshes, polynomial patches, implicit surfaces, etc. Tools used to manipulate those surface representations have evolved to meet a broad spectrum of modeling needs, from high precision design to fast prototyping.

We introduce a modeling primitive to quickly and intuitively produce objects with good properties for its surface and its topology. We are interested in a modeling primitive that can be used in two modeling contexts:

- Interactive modeling: fast and intuitive construction of an approximate object, that can be subsequently easily sculpted and modified by a user.
- Procedural modeling: easy topology specification, volumetric definition (for CSG operations), general surface parameterization, and good surface control.

1.1 Related work

When modeling objects with polygons, subdivision surfaces [1], and polynomial patches [16], the artist must be very careful to avoid self-intersections, cracks, duplicated vertices, incoherent interior/exterior definition, discontinuous surface parameterization, etc. These problems are accentuated when the resulting objects must have consistent surface and volume properties.

Implicit surfaces [4] and F-Rep [15] offer continuous surfaces with valid interior/exterior properties. Unfortunately their limit surface can be complex to extract, and a good surface parameterization can be difficult to provide due to, among a number of problems, changes in topology.

ZSpheres/*B-Mesh* [10, 17] form a very flexible modeling primitive based on a subdivision mesh enclosing a tree of spheres. It offers good surface and volume properties and a consistent surface parameterization, allowing one to sculpt surface details through displacement mapping. It is well designed for organic-like objects, but less for CAD-like objects with sharp edges.

L. Leblanc (✉) · J. Houle · P. Poulin
LIGUM, Department I.R.O., Université de Montréal, Montreal, Canada
e-mail: leblanc@iro.umontreal.ca

J. Houle
e-mail: houlejo@iro.umontreal.ca

P. Poulin
e-mail: poulin@iro.umontreal.ca

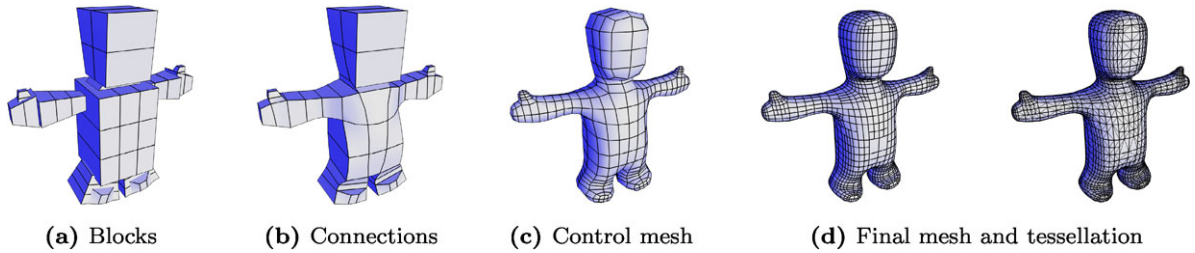


Fig. 1 The four stages of the pipeline for modeling with blocks

Polycube maps [19, 21] have been used to efficiently represent objects of different topologies, but less as a complete modeling primitive. Each face of a polycube encodes well a displacement map to generate the final polygonal mesh.

1.2 Overview

Inspired by these last two representations and by implicit surfaces in general, we have integrated a number of their key concepts into our block modeling. Each block can be interpreted as a cube in a polycube map, with its associated parameterization of faces. Blocks are assembled by connecting faces, similar to implicit surfaces and *ZSpheres*, but the connection is controlled with a resolution for each block face. The resulting connected blocks provide the basic shape of an object. The parameterization of each exterior block face is used to encode surface details, pasted on top of an adaptive subdivision surface. The original block edges also provide a mechanism to generate sharp and smooth edges on the final surface.

This combination of representations leads to an intuitive, easy to control, and general modeling tool that can generate a wide variety of objects with consistent surface and volume properties, and of different topologies.

The paper is organized as follows. First, we describe in Sect. 2 our basic block primitive, connections between blocks, adaptive meshing, the usage of constructive solid geometry (CSG) operations, and some of our experience about modeling with blocks. Then we present and discuss features for some typical results in Sect. 3. We finally compare our technique with other more closely related modeling schemes in Sect. 4, before concluding and discussing extensions in Sects. 5 and 6.

2 Modeling with blocks

The strategy behind modeling with our blocks is to first build a coarse shape with the correct topology, without having to deal with the details of low-level topology operations or description. Then this coarse shape is refined with a displacement function. This is beneficial to both interactive modeling and procedural modeling.

An object built with blocks is defined by three components. First, a set of blocks (our main primitive) is used to describe the main parts of the object. Then, links specify connectivity between each block. Together, they define the control mesh with correct topology. Finally, the simple parameterization inherited from the blocks can support an optional displacement to generate the final result.

From an implementation point of view, the pipeline to generate an object is divided into four stages: definition of blocks, connection between blocks, creation of the control mesh, and generation of the mesh. An illustration of this pipeline appears in Fig. 1.

2.1 Blocks

A *block* is a volumetric primitive akin to a cuboid defined by eight vertices and six faces. There are no restrictions on the positions of the vertices, except that they should generate a valid interior, consisting of one continuous 3D space. This is however not enforced in our system.

Each face of a block is defined as a bilinear patch that is divided independently of its adjacent faces into a regular grid of sub-faces of any resolution. Faces and sub-faces are strictly quadrilaterals; they need not be planar.

The sub-faces are the elements defining the geometry. They can be connected together (Sect. 2.2) to form the surface using Catmull–Clark subdivision [7], inheriting its properties: C^2 continuous everywhere except C^1 for vertices of valence $\neq 4$.

To alter smoothness, each edge of a block face can be tagged as sharp. Figures 2 and 3 show different definitions for a block with sharp and smooth edges, along with their resulting geometries. Figure 4 shows the relations between faces, sub-faces, patches, and sub-patches.

Now that we have a better understanding of the structure for a single block, we will see in the next sections how blocks can connect into groups of blocks.

2.2 Connections

Once blocks are fully defined (face subdivision and edge sharpness) and positioned to establish the coarse shape of

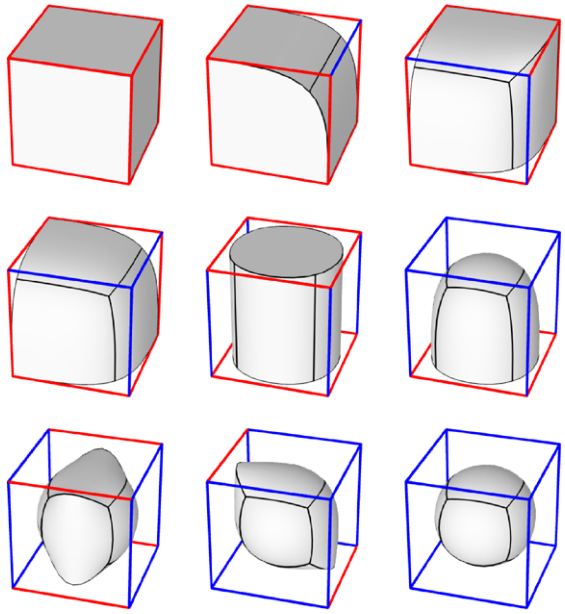


Fig. 2 Different configurations of smooth/sharp edges in a block. *Red block* edges produce sharp mesh edges, *blue block* edges produce smooth mesh edges

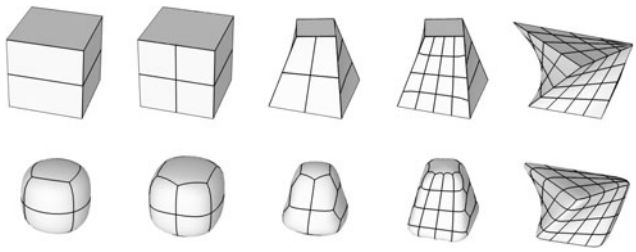
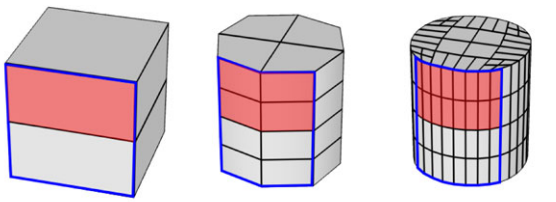


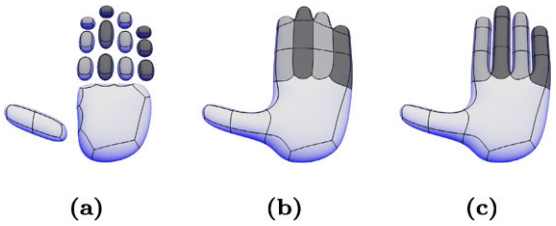
Fig. 3 *Top row*: increasing the number of sub-faces and moving vertices in a block with sharp edges. *Bottom row*: same configurations with smooth edges



(a) Faces and sub-faces **(b)** Patches **(c)** Sub-patches

Fig. 4 The four elements of the block primitive. In the specific example of this figure: (**a**) One face is divided into two sub-faces. (**b**) One sub-face is subdivided into four patches. (**c**) One patch is subdivided into four sub-patches. *The blue contours* indicate regions associated with one original face of the block, and *filled red* indicates regions associated with one sub-face

the desired object, connections can be computed between their sub-faces. Those connections, between pairs of faces or sub-faces of neighboring blocks, automatically create the



(a) **(b)** **(c)**

Fig. 5 Three variations of a hand where shades represent three group IDs: (**a**) no connections, not even within the same group ID; (**b**) all groups connect together; and (**c**) group IDs of fingers connect to the hand, but not between each pair of adjacent fingers. To connect with the fingers, the palm's top face is subdivided into 4×1 sub-faces and the left side into 3×1 sub-faces (the thumb connects to the middle sub-face)

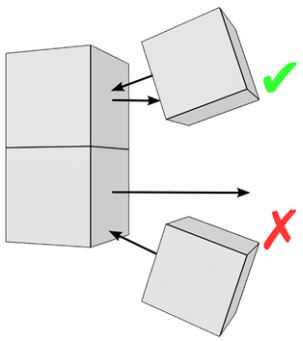


Fig. 6 Sub-faces connection. The upper sub-faces connect together since they are each other's closest sub-face, in contrast to the lower pair

topology of the object. This process requires only a small set of attributes assigned by the user: a group ID for every block, a global list of group ID pairs, and a scalar distance threshold. The group ID pairs define which parts of the model are allowed to merge together (see Fig. 5). The threshold value determines the maximum allowed connection distance; it is scale invariant by considering the perimeters of both sub-faces A and B:

$$\text{distance} < \text{threshold} \times \big(\text{perimeter}(A) + \text{perimeter}(B)\big).$$

Automatic connections work as follows. For every sub-face (or the whole face, if it is not subdivided) of every block, a ray is cast from the center of the sub-face outwards along the normal (see Fig. 6). A connection between sub-face A (caster) and sub-face B (nearest hit) exists if and only if:

1. Sub-faces A and B belong to different blocks.
2. The group ID of A is allowed to link with the group ID of B (from the global pair list).
3. The distance between A and B is within the specified threshold.
4. Sub-face B is A's closest sub-face, and vice versa.
5. No degenerate edges are created (Fig. 7).

To detect connections, simple ray-casting is used from the sub-face's center position. This has the advantage of being fast, easy to implement, and nonambiguous. If this is

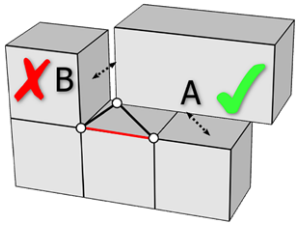


Fig. 7 Example of an invalid connection. If connection B is executed after A has already been connected, the three vertices will merge and form a degenerate edge (*red*)

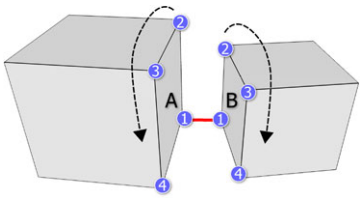


Fig. 8 Connection between two sub-faces. The vertices numbered 1 form the starting connection pair

deemed too limiting, it can be replaced with any coverage computing technique such as casting multiple rays or a form of shaft-tracing.

To prevent inconsistent topology (condition 5, above), only one connection can be established between two blocks. As such, only the connection with the shortest distance will be kept. A list of manual connections, independent of the actual distances, can be specified for added flexibility by specifying the two sub-faces to be joined for each connection.

After all connections have been established, vertex positions for all connected sub-faces are modified to be joined together. To do so, we first determine a relation between pairs of vertices from both sub-faces. Since we work on quadrilaterals, there are four possible ways of connecting the two sub-faces. To find the best one, we first choose an arbitrary pair of vertices, one vertex from each sub-face, then we match the remaining vertices in counterclockwise order for sub-face A and clockwise order for sub-face B. Figure 8 illustrates this process. We compute the total distance between each pair of vertices as the cost of the connection. We test the other three configurations by changing one vertex of the starting pair to each of the other vertices of the sub-face. After evaluating all these connections, we select the configuration having the lowest cost.

The new position of joined vertices is computed as their average. It is important to note that more than two vertices can be joined together, notably, for vertices lying on the edges of a block with multiple neighbors (see Fig. 9). In that case, the resulting vertex can have any valence higher than 2, depending on the number of connected blocks. Nothing prevents us from computing a weighted average of the positions. It is however unclear how those weights can automatically or intuitively be set up, especially in cases of multiple connections, such as those shown in Fig. 9.

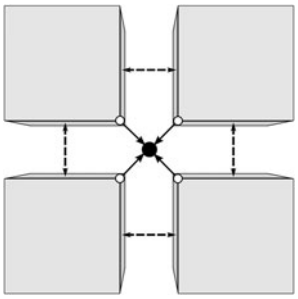


Fig. 9 Computation of a joined vertex of valence 4. *Dashed segments* represent connections between blocks. *The center black dot* is the new vertex position computed as the average of four vertices

2.3 Control mesh

The next step creates a control mesh to generate a subdivision surface. This control mesh is created by assembling the set of all exterior sub-faces (i.e., sub-faces that are not connected) in a watertight mesh. Interior sub-faces are simply ignored and do not participate in the final geometry. An edge is tagged as sharp (i.e., forming a crease) when at least one of the original block edges forming this edge is sharp. Priority is given to sharpness over smoothness since sharpness is an added property on an edge.

The assembled exterior sub-faces form a mesh of quadrilaterals, possibly containing T-vertices due to different resolutions for the grid of sub-faces. This is corrected in two steps. First, vertices are added to quadrilaterals containing T-vertices, transforming them into n-gons. Second, one pass of Catmull–Clark subdivision [7] is applied to obtain a control mesh composed exclusively of quadrilaterals.

2.4 Mesh generation

The last step of the process uses subdivision surface to generate the final mesh. The control mesh made exclusively of quadrilaterals (Fig. 1) is adaptively subdivided with respect to both local curvature and applied displacement of the model. Boolean operations are supported by inserting edges and vertices at interpenetrating locations and determining interior and exterior regions through the evaluation of the CSG tree. A final meshing operation yields a closed (watertight) triangular mesh.

2.4.1 Patch subdivision

To achieve a high quality mesh with fewer subdivisions, we approximate the subdivision surface with parametric patches [14]. This has two main advantages over conventional Catmull–Clark subdivision: at any subdivision level, each vertex is immediately positioned on its limit surface, and the subdivision pattern is decoupled from the evaluation.

After converting each quadrilateral of the control mesh to a parametric patch, we subdivide the patch into a set of sub-patches in a kd-tree manner [11]. Each vertex of the

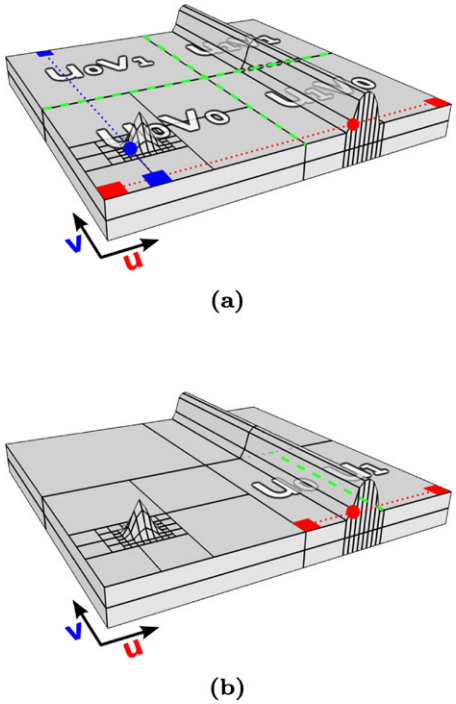


Fig. 10 A patch is subdivided in u when a 3D point from a texel in a row is farther than a certain threshold from the segment formed by the first and last texels; similarly in v. *Top*: subdivision of a patch in uv. *Bottom*: subdivision of a sub-patch in u only

sub-patch is placed at the location derived from its parametric patch evaluation, and moved by its displacement map or function. The parameterization used is similar to the one by Burley and Lacewell [5], where each patch has its unique mapping of size related to its world size or its level of detail.

A sub-patch is subdivided according to the following metric. If the displacement is expressed as a function (and not as a map), we build a geometry image [8] of the desired quality, thus converting the displacement function as a map. We only need to keep a small number of patches in memory, i.e., the current patch and its neighbors, in order to blend edges and avoid cracks in the displacement function.

A sub-patch is either subdivided in u, in v, or in both uv directions. In each case, the sub-patch is subdivided at its middle position. A two-pass evaluation of the applied displacement map is done, one in u and the other in v. When evaluating in u, we compare each texel (a displaced 3D point), row by row, to the 3D segment defined by the first and last texels of the row. As soon as one texel is farther than a specified threshold, the sub-patch is marked to be subdivided in u. The evaluation in v is similar, but is done column by column. Figure 10 illustrates this process. In the top figure, the red dot indicates that a subdivision is requested in u when traversing the rows of texels, and the blue dot in v. The sub-patch is therefore subdivided in four sub-patches labeled $u_i v_j$ in gray. In the bottom figure, for one sub-patch, the red dot indicates that a subdivision is requested in u,

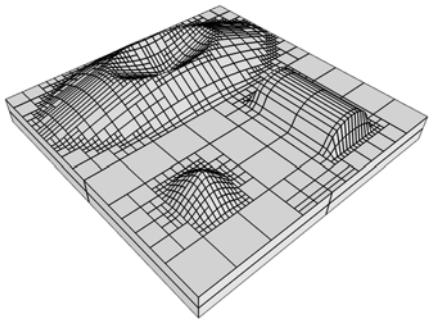


Fig. 11 A patch with an associated displacement map is subdivided hierarchically and anisotropically into sub-patches

but no such subdivision is requested in v. This sub-patch is therefore subdivided only in u into two sub-patches. This metric provides a good anisotropic subdivision scheme, as can be observed in Fig. 11.

2.4.2 CSG

Since our blocks give a volumetric definition, CSG evaluation can be used to increase the expressiveness of an object. The boolean operations are expressed in a complete CSG tree, supporting the standard operations (union, subtraction, intersection) where each leaf is a group of blocks. After the adaptive subdivision of the sub-patches, linearization of all the edges of the sub-patches is performed to simplify the intersection computation (a patch can now be represented by two triangles without creating cracks).

The process starts by finding which patches intersect each other. For each pair of intersecting sub-patches, segments at the intersections are created and assigned to both sub-patches (Fig. 12b for one pair). Those segments are generated by approximating each sub-patch with a pair of triangles, and performing a triangle-triangle intersection (Fig. 12a). After computing all intersections, all the segments associated with one sub-patch are intersected between themselves. An intersected segment is divided into two sub-segments at the intersection point. The resulting segments are then traversed to form nonoverlapping polyline loops (Fig. 12c). For each of those loops, a ray is cast along the normal from a point contained within the loop. This ray is intersected with all the other patches and compared to the CSG tree in order to determine if the surface bounded by the loop is on the final mesh (after CSG). If a loop is on the surface, it is tagged as such, and so are its sub-patch and its patch. This way, when tessellating the object, entire patches or sub-patches can be skipped if not tagged.

2.4.3 Meshing

While doing patch subdivision, vertices are inserted on all neighboring sub-patches sharing an edge in order to avoid

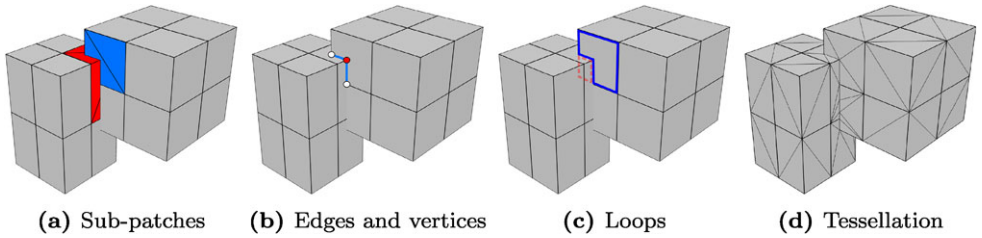


Fig. 12 The different stages of a CSG union evaluation. (**a**) Intersection test of *the blue sub-patch* against *the red sub-patches*. (**b**) Insertion of intersecting edges and vertices. (**c**) Intersection of the segments on one sub-patch, stitching into closed loops, and testing against the CSG tree. (**d**) Tessellation of the union of the two blocks, without any interior geometry

cracks due to T-vertices. If CSG was not used, or if a sub-patch contains only one loop (the sub-patch contour), a simple triangulation algorithm such as the one of Cignoni et al. [6] can be used. In cases of a complex concave boundary or multiple loops, we use an ear-clipping algorithm [9].

2.5 Usage and limitations

Our experience shows that the process of describing objects with blocks is fast and fairly easy. There are no restrictions on the possible topologies that can be reproduced, when proceeding by first building a coarse shape, and then adding details with displacement mapping. While we have not encountered any problem so far, we have also not tried to build many precise control meshes using only blocks. However, we can speculate that some cases could prove more difficult.

For example, when dealing with a number of blocks of different scales, it could be a limiting factor that each face has only its own fixed grid of sub-faces. This could result in the need to split a block into smaller blocks. Allowing the subdivision of sub-faces into multi-level grids could help in such cases.

Also, in some cases with lots of connections, setting correct group IDs to ensure that the resulting connections are the desired ones, could prove a little tedious. However, this should still be a lot easier than manually handling the issues due to topology.

3 Results

Figures 13–20 show results modeled exclusively with blocks. They illustrate the flexibility of the block primitive as it is used to model architectural objects, such as buildings and staircases, as well as more organic shapes, such as trees and characters. Objects containing both a blend of sharp and smooth edges, such as a chair, are also well adapted to modeling with blocks.

All buildings in Figs. 18, 19 and 20 and the staircases in Fig. 17 are modeled procedurally with our in-house system [13]. This procedural system uses blocks as its foundation to generate the geometry, and as a result, gains several

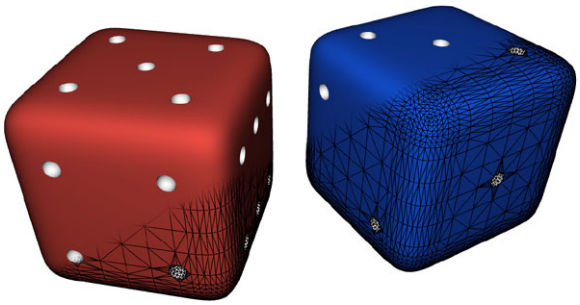


Fig. 13 The general shape of each die is modeled by one block with a 4×4 subdivision for each face, and all smooth edges. Each dot is modeled as a small block with no sub-faces, and all smooth edges. Each dot is subtracted from the general die shape. *The black segments* show the final tessellation

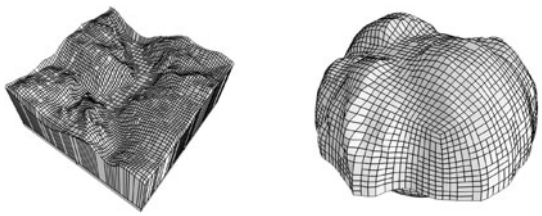


Fig. 14 Terrain (*height field*) displaced on a sharp flat face of a block, and on a smooth rounded face. Sub-patches are drawn as *black segments*

interesting features: every entity has a volume (i.e., walls are not paper-thin), and the tessellation is watertight and adaptive. Moreover, CSG enables easy insertion of doors and windows to existing walls by first subtracting blocks from the wall to create a hole, and then adding the window or door. Connections ensure that no cracks are left out between a window and its pierced wall. To support physical simulation, we could easily extend our block description to include material properties.

4 Comparing with other modeling schemes

Many modeling primitives and techniques exist, each having its own set of advantages and disadvantages. Among

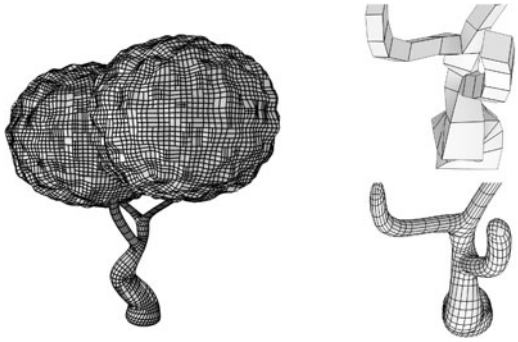


Fig. 15 Stylized tree. The twisting trunk is modeled by ten blocks, each branch by five to seven blocks, and the foliage by two large blocks. All edges are smooth. The block connecting the two top branches forming a Y-intersection has its top face subdivided in 2×1. *The images on the right* show a close-up view on the connecting branches where *the top one* shows the blocks, and *the bottom one* the resulting sub-patches. The object consists of 30 blocks, 502 patches, 13,498 sub-patches, and 25,778 triangles. Sub-faces are drawn as *black segments*

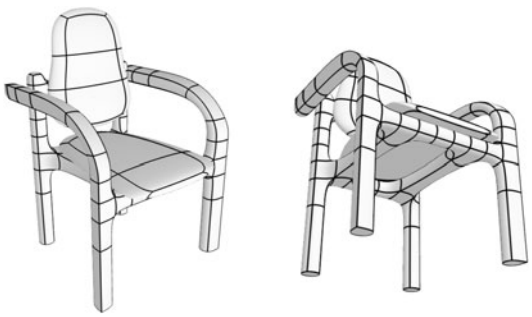


Fig. 16 Two views of an office chair. The bottom of each leg is modeled with sharp edges, as well as one edge for each leg raising up to become an arm. Sub-faces are drawn as *black segments*

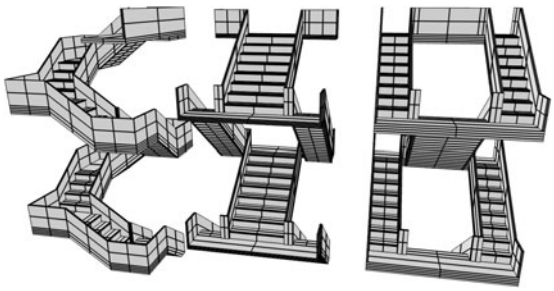


Fig. 17 Procedural staircases modeled with blocks and sharp edges. Patches are drawn as *black segments*

them, the implicit surfaces such as the *blobtree* [20] and the *ZSpheres* [17] are closer to our block primitive.

4.1 Implicit surfaces

Implicit surfaces, also called *metaballs* or *blobbies*, were introduced by Blinn [3] as the isosurface of a field function defined by the sum of simple primitive functions such as the

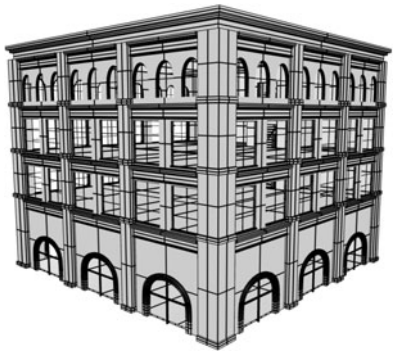


Fig. 18 The four-story office building has mainly empty interior spaces, except for a staircase. It consists of 1,694 blocks, 34,864 patches, 62,836 sub-patches, and 105,264 triangles

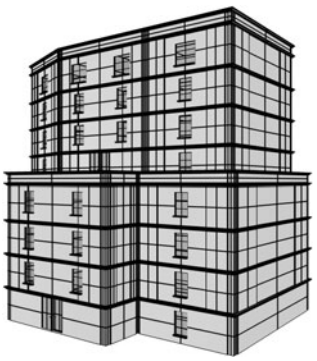


Fig. 19 The hotel building has a mid-level terrace, and unfurnished rooms. It has 986 blocks, 22,832 patches, 23,408 sub-patches, and 36,658 triangles

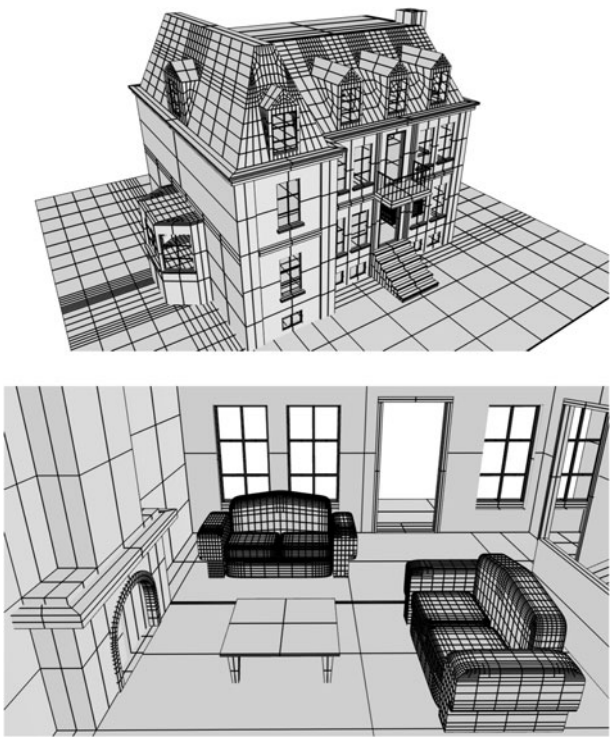


Fig. 20 *The bottom image* is an interior view from the three-story house *in the top image*. It has 1,407 blocks, 28,456 patches, 33,036 sub-patches, and 58,632 triangles. For all the buildings, all windows and doors result from CSG operations

Gaussian distance from a point. In the *blobtree* [20], the density field is described as a tree of operations, and it supports blending, warping, and boolean operations. To better control the locality of the blending, Bernhardt et al. [2] present a novel solution limiting its range to the intersection of the primitives.

We can think of our block connections as a discrete form of *blobbies*. Compared to them, our connections are less versatile, because *blobbies* can connect anywhere, no matter what the position of the basic primitives (sphere, skeletal, etc.) is, in contrast to blocks, where a connection is made only sub-face to sub-face. However, with blocks, we have much more control over the final appearance of the surface and can easily add sharp edges. Our subdivision surface leads to better tessellation and the surface has a good parameterization, an aspect which *blobbies* sorely lack.

4.2 ZSpheres

ZSpheres [17] and *B-Mesh* [10] (a variant of the former) describe an object as a hierarchy of spheres. A control mesh is built on top of the spheres, keeping the same topology as the tree. Unlike our blocks, *ZSpheres* do not seem to support complex topology (genus > 0), but should be modifiable to do so. They are only suitable for smooth, organic-like objects, and cannot easily model architectural or mechanical objects. Also, they cannot model complex surfaces containing creases, valleys, and ridges without displacement mapping. Since their mapping is done automatically based on the control mesh, they could prove difficult to use in a procedural context.

It should be noted that our block approach could emulate *ZSpheres* results by using a similar tree description.

5 Conclusion

We have presented a simple block primitive to easily model objects in both interactive and procedural contexts. Our block primitive possesses important and desirable characteristics:

- Simple topology specification with connections
- Valid volumetric definition
- Good control over the surface with editable block vertices
- Adaptive surface meshing with the subdivision surface

The block representation is fairly compact, considering the number of final triangles that can be generated. In the buildings of Figs. 18, 19, and 20, consisting mainly of flat surfaces, one block generates between 37 to 62 times more triangles. For the tree of Fig. 15 with surfaces that are more curved, one block generates on average close to 860 times more triangles. While this is clearly related to the subdivision metric, we have been conservative with respect to the obtained visual quality.

We consider that our modeling system represents a good step in the direction of defining a simple, yet powerful, modeling primitive.

6 Future work

There are several interesting avenues that we propose to explore. For connections, weights could be added, faces could be subdivided into sub-faces automatically, and another type of connection could be permitted by filling the space between two blocks instead of merging their vertices. In the latter case, it could be generalized to allow more than a two-block connection. T-Splines [18] could possibly be used as a replacement for the Catmull–Clark subdivision surface, thus reducing the tessellation and distortions of the parameterization in some cases.

Compared to polycube maps [19, 21], a block model with its more flexible vertex configurations should represent an object with fewer cubes (blocks) while more closely matching the shape. It would therefore offer a better compression of the associated displacement maps.

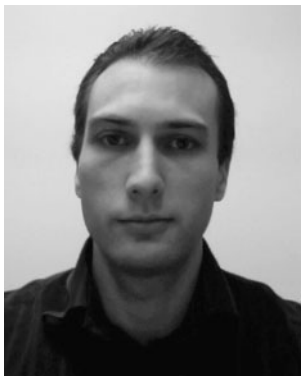
In this paper, we presented a tessellation algorithm for the blocks, but since blocks are higher level primitives, they can be converted to different formats. Converting to voxels with an algorithm based on work by Lai and Chang [12] should enable faster and more robust CSG operations.

Acknowledgements The authors thank the anonymous reviewers for their constructive comments, and acknowledge financial support from FQRNT, NSERC, and GRAND.

References

1. Andersson, L.E., Stewart, N.F.: Introduction to the Mathematics of Subdivision Surfaces. SIAM, Philadelphia (2010)
2. Bernhardt, A., Barthe, L., Cani, M.P., Wyvill, B.: Implicit blending revisited. Comput. Graph. Forum **29**(2), 367–375 (2010)
3. Blinn, J.F.: A generalization of algebraic surface drawing. ACM Trans. Graph. **1**(3), 235–256 (1982)
4. Bloomenthal, J. (ed.): Introduction to Implicit Surfaces. Morgan Kaufmann, San Mateo (1997)
5. Burley, B., Lacewell, D.: Ptex: per-face texture mapping for production rendering. In: Eurographics Symposium on Rendering '08, pp. 1155–1164 (2008)
6. Cignoni, P., Montani, C., Scopigno, R.: Triangulating convex polygons having T-vertices. J. Graph. GPU Game Tools **1**(2), 1–4 (1996)
7. DeRose, T., Kass, M., Truong, T.: Subdivision surfaces in character animation. In: SIGGRAPH '98, pp. 85–94 (1998)
8. Gu, X., Gortler, S.J., Hoppe, H.: Geometry images. In: SIGGRAPH '02, pp. 355–361 (2002)
9. Held, M.: FIST: fast industrial-strength triangulation of polygons. Algorithmica **30**(4), 563–596 (2001)

10. Ji, Z., Liu, L., Wang, Y.: B-mesh: a modeling system for base meshes of 3D articulated shapes. Comput. Graph. Forum, Proc. Pac. Graph. **29**(7), 2169–2178 (2010)
11. Lai, S., Cheng, F.: Adaptive rendering of Catmull–Clark subdivision surfaces. In: CAD-CG '05: Proc. Intl. Conf. Computer Aided Design and Computer Graphics, pp. 125–132 (2005)
12. Lai, S., Cheng, F.: Voxelization of free-form solids using Catmull–Clark subdivision surfaces. In: GMP'06: Lecture Notes in Computer Science, pp. 595–601. Springer, Berlin (2006)
13. Leblanc, L., Houle, J., Poulin, P.: Component-based modeling of complete buildings. In: Graphics Interface 2011 (2011)
14. Ni, T., Yeo, Y., Myles, A., Goel, V., Peters, J.: GPU smoothing of quad meshes. In: SMI'08: IEEE Intl. Conf. on Shape Modeling and Applications, pp. 3–9 (2008)
15. Pasko, A., Adzhiev, V., Sourin, A., Savchenko, V.: Function representation in geometric modeling: concepts, implementation and applications. Vis. Comput. **11**, 429–446 (1995)
16. Piegl, L., Tiller, W.: The NURBS Book. Springer, Berlin (1995)
17. PIXOLOGIC: ZBrush (2011). http://www.pixologic.com/
18. Sederberg, T.W., Zheng, J., Bakenov, A., Nasri, A.: T-splines and T-NURCCs. ACM Trans. Graph. **22**, 477–484 (2003)
19. Tarini, M., Hormann, K., Cignoni, P., Montani, C.: Polycube-maps. In: SIGGRAPH '04, pp. 853–860 (2004)
20. Wyvill, B., Galin, E., Guy, A.: Extending the CSG tree. Warping, blending and boolean operations in an implicit surface modeling system. Comput. Graph. Forum **18**(2), 149–158 (1999)
21. Xia, J., Garcia, I., He, Y., Xin, S.Q., Patow, G.: Editable polycube map for GPU-based subdivision surfaces. In: I3D '11: ACM Symposium on Interactive 3D Graphics and Games, pp. 151–158 (2011)



Luc Leblanc is a Ph.D. candidate in Computer Graphics at the Universite de Montreal. After completing his M.Sc. degree at the Universite de Montreal, he worked for four years at Autodesk in Montreal. His research interests are in procedural modeling, real-time rendering, illumination, and animation.

Jocelyn Houle completed his M.Sc. in Computer Graphics at the Université de Montréal. He later worked as a hardware engineer at ATI, specializing in texturing; he was instrumental in the texture pipe design of both the Xbox 360 project and the R600 GPUs. He is the co-founder of Ludo Sapiens, a game technology company.

Pierre Poulin is a full professor in the Computer Science and Operations Research department of the Universite de Montreal. He holds a Ph.D. from the University of British Columbia and an M.Sc. from the University of Toronto, both in Computer Science. He has served on program committees of more than 40 international conferences. His research interests cover a wide range of topics, including image synthesis, image-based modeling, procedural modeling, natural phenomena, scientific visualization, and computer animation.