

Building 3D Models Using Plastic Construction Bricks

Nathan Cournia, Karl Rasche and Andrew Van Pernis

{acnatha, rkarl, arakel}@vr.clemson.edu.

Clemson University

-
-
-



Outline



-
-
-
-
-
-
-
-
-

Outline

- Motivation
- Representing 3D Models As Voxels
- Representing 2D Images As Voxels
- Converting Voxels To Plastic Construction Bricks
- Results



Motivation

Problem Definition

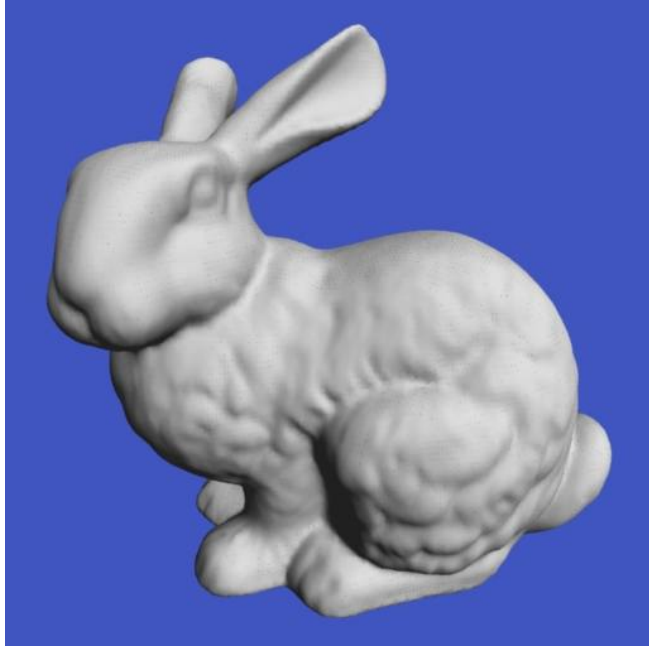
Problem: Given a three-dimensional model how can we build that model using plastic construction bricks?

Problem Definition

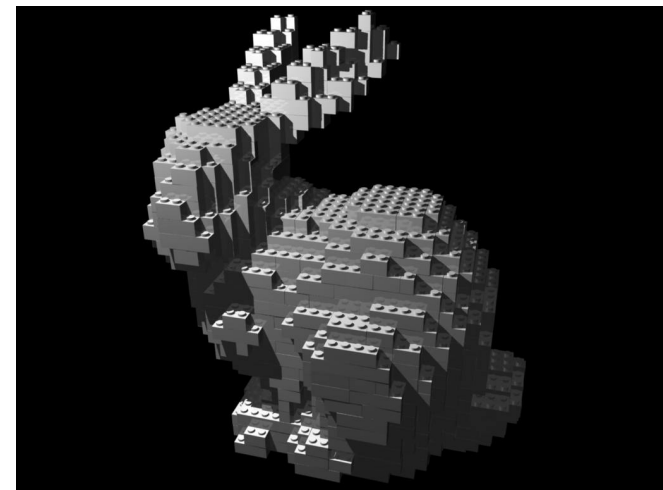
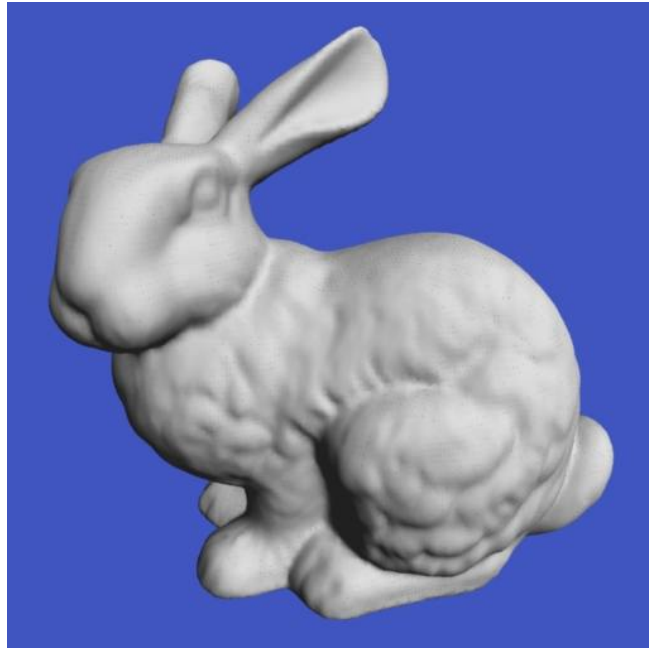
Problem: Given a three-dimensional model how can we build that model using plastic construction bricks?

Our answer: Create a voxelization of the model, then represent that voxelization as a set of LDraw parts which have been carefully selected to produce a build-able version of the model.

Problem In Pictures



Problem In Pictures



Motivation

- Artistic expression
 - Mosaics
 - Sculptures
 - Music videos
 - Animations

Motivation

- Artistic expression
 - Mosaics
 - Sculptures
 - Music videos
 - Animations
- Technical idea + creativity \Rightarrow publication

Motivation

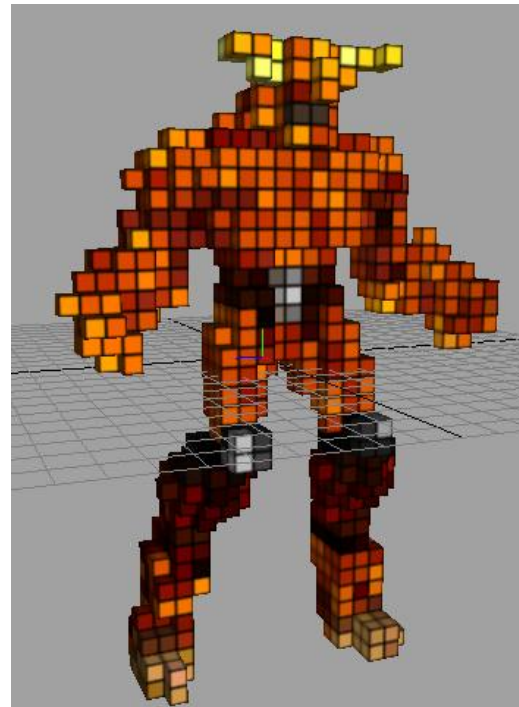
- Artistic expression
 - Mosaics
 - Sculptures
 - Music videos
 - Animations
- Technical idea + creativity \Rightarrow publication
- Block construction turns out to be an interesting problem



Representing 3D Models As Voxels

Voxelization

- Given a polygonal model, generate a set of voxels



What is a voxel?

- A small 3D box with some properties (density, color, etc)
- Short for volume element
- 3D counterpart of a 2D pixel

Voxel Visualization

- Can view voxel data as slices
- Viewing voxel data is useful for many applications, specifically in medicine

See http://www.nlm.nih.gov/research/visible/visible_human.html

Voxelization

- Process of converting a geometric representation of a synthetic model into a set of voxels
- Some processes are easier to use when operating on voxel data
 - Computing Volume
 - CSG (Intersection, Union, etc)
 - Collision Detection
- The reverse is also true; Voxels are not optimal for everything

Voxelization

- Several methods for generating a voxelization exist
 - Casting Rays
 - Scan Converting
- Most are not pretty to code
- Can be slow
- Can we speed it up? Yes, use the GPU

See <http://www.cs.sunysb.edu/vislab/projects/volume/Papers/Voxel/>

Hardware Accelerated Voxelization

- Modern graphics hardware is fast (exceeding Moore's Law)
- Modern graphics hardware is programmable
- Computationally expensive algorithms are being offloaded to the GPU
 - Fast Fourier Transform
 - Global Illumination Computations
 - Path Finding (AI)
 - Collision Detection

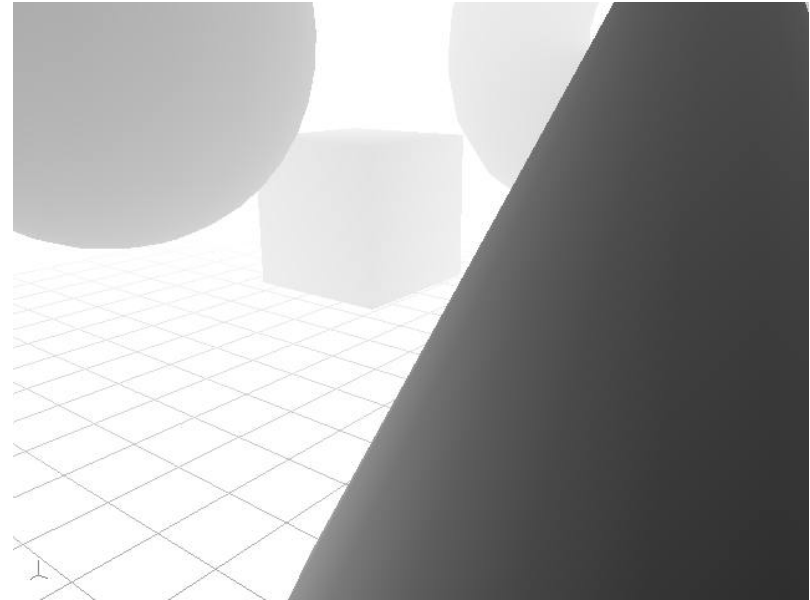
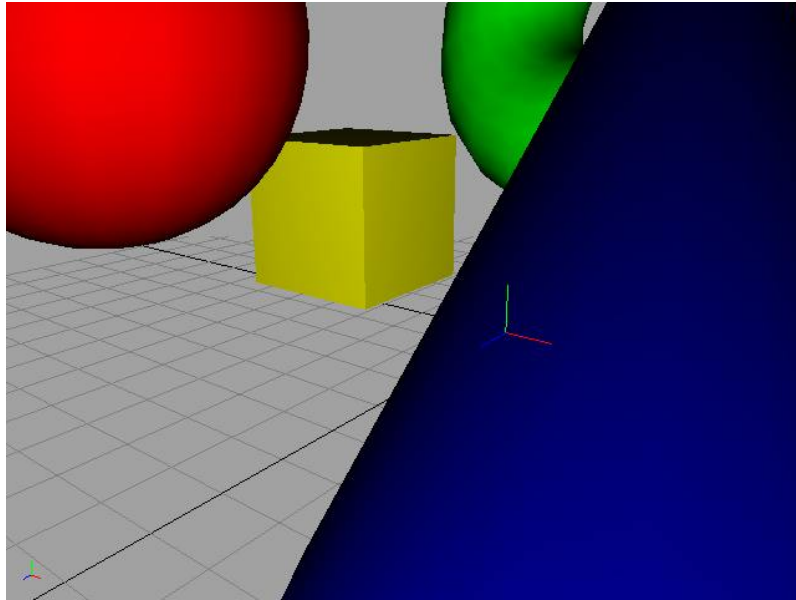
Hardware Accelerated Voxelization

- Turns out we don't need to use the programmable GPU
- Standard fixed function color and depth buffers will suffice
- Color buffer stores the color of each pixel in the rendered scene
- But what is the depth buffer?

Depth Buffer

- Also known as the Z-buffer
- Stores the depth of each pixel in the scene
- Values range from $[0,1]$
- Usually initialized to 1
- Smaller values are closer to the near clip plane (camera)
- Traditionally used for hidden surface removal

Depth Buffer Example

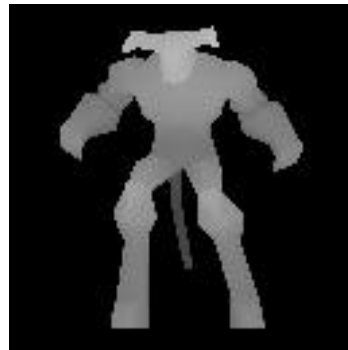


HW Voxelization Algorithm

- Center mesh around world origin
- Set viewport to $N \times N$, where N is the size of a dimension in the $N \times N \times N$ voxel lattice
- Render scene through a tightly fitted orthographic (parallel) projection from each side of the model's bounding box
- Record depth and color buffers from each render
- Produces six set of images (6 color buffer images, 6 depth buffer images)

Depth/Color Buffer Visualization

- Front/Back



Depth/Color Buffer Visualization

- Left/Right



Depth/Color Buffer Visualization

- Top/Bottom



Voxelization Cube



Determining if a Voxel is Filled

- For each voxel in the lattice, compute the distance to each side of the cube
- Test if this distance is bounded by the depth buffers in each axis
- If so, voxel is considered filled, otherwise, it's empty
- Holes not visible from outside the object will be filled

Determining a Voxel's Color

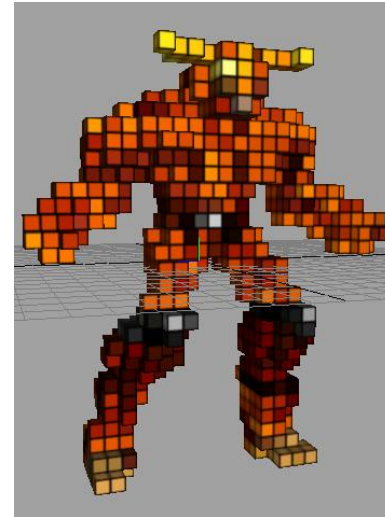
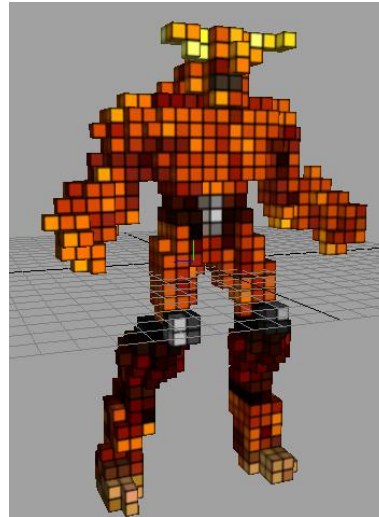
- Find the depth map which is closest to the voxel
- Look up the voxel's color in the depth map's corresponding color map
- Interior voxels share the same color as closest surface voxel

Voxelization Algorithm Notes

- Works for polygonal and analytical models
- OpenGL display lists can be used to improve rendering time for complex models
- Bounding cube needs to be slightly larger than model

Plastic Construction Brick Specifics

- Plastic construction bricks are not cubes
- A 1×1 brick has a $\frac{5}{6}$ aspect ratio
- Model must be squashed during voxelization to preserve aspect ratio



Demo

Representing 2D Images As Voxels

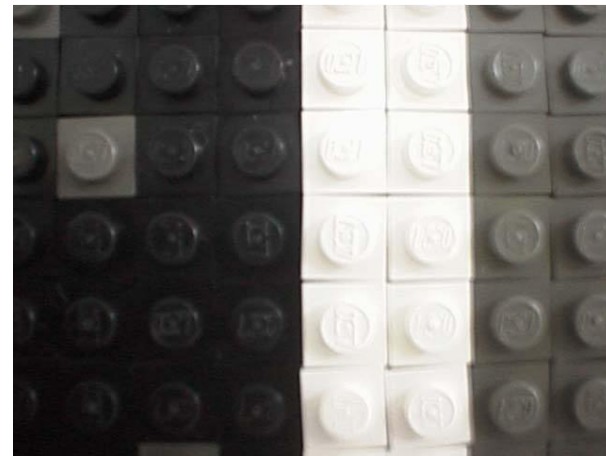
Voxelization of Images

- Goal: Given any 2D image, output a voxelization
- Pixels directly correspond to a voxel slice
- Perform image scaling to reduce/enlarge to desired number of voxels

Brick Mosaics



Studs-up

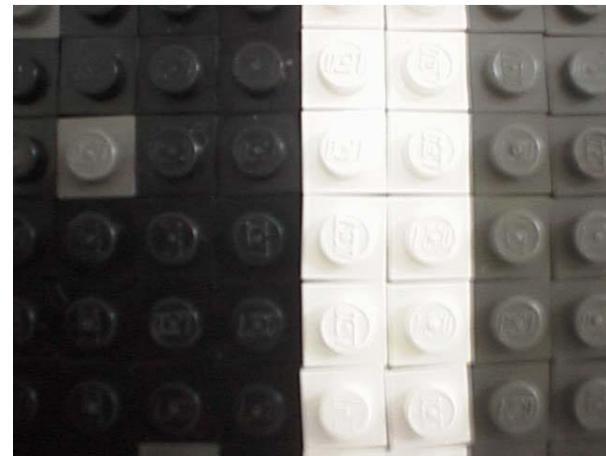


Studs-out

Brick Mosaics



Studs-up
X-Y plane



Studs-out
X-Z plane

Giving Voxels Color

- Plastic construction bricks have a limited number of colors
- Convert image to brick palette
- Problem: Limited number of colors in brick palette causes banding



Converting to Brick Palette

- Solution: Dither the image
- Dithering is the addition of a sub-quantum signal (often high frequency noise) to a signal of interest that is being quantized
- Many dithering algorithms exist

Dithering

- We use Floyd-Steinberg dithering
- Spreads error in quantization over neighboring pixels



Generating a Construction Plan

- We use a simple greedy algorithm that processes 1 brick row at a time





Converting Voxels To Plastic Construction Bricks

What Is LDraw?

- Originally, DOS programs LDraw and LEdit with a file format representing several different plastic construction brick shapes
- Many editors have been developed using the LDraw parts file format
- Programs exist to convert to a variety of 3D applications
- Maintained by a standards committee now

LDraw File Format

- ASCII text
- First value is an integer indicating line type

#	Line Type
0	comment or meta-command
1	reference to another LDraw file
2	line between two points
3	triangle
4	quadrilateral
5	conditional line between two points

LDraw File Format (cont.)

- Reference line type is as follows:
1 <color> <transformation matrix> <file name>
- Line between two points is as follows:
2 <color> <point1> <point2>
- Triangle line type is as follows:
3 <color> <point1> <point2> <point3>
- Quadrilateral line type is as follows:
4 <color> <point1> <point2> <point3> <point4>

LDraw Conventions

Brick Sizes - given using a
Width×Length×Height notation

Part Numbers - each different brick is given a
unique part number; attempts to match those
used by the LEGO™ Corporation

Colors - based on those used by the LEGO™
Corporation; each is assigned a unique
integer

The Simple Answer

Replace each voxel with a 1x1 brick of the appropriate color. Interior voxels can be assigned any color desired.

The Simple Answer

Replace each voxel with a 1x1 brick of the appropriate color. Interior voxels can be assigned any color desired.

Problem: This solution does not create a build-able version of the model.

Algorithm Goals

- Replace neighboring voxels of the same color with an appropriate brick
- Check for support for a brick in the layer above or below
- Use $2 \times n$ bricks whenever possible
- Avoid 1×1 bricks whenever possible
- Fix building impossibilities

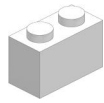
A Greedy Algorithm

- Voxelization is of size $x_{size} \times y_{size} \times z_{size}$
- Consider only an X-Z slice of the voxelization
- Search for adjacent voxels in the X direction first
- Search for adjacent voxels in the Z direction second
- Replace neighboring voxels with the largest brick possible

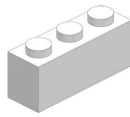
A Greedy Algorithm's Bricks



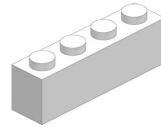
1×1



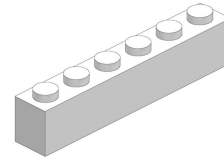
1×2



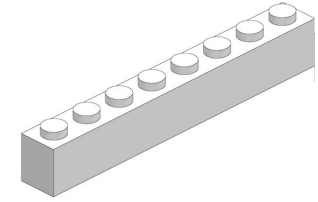
1×3



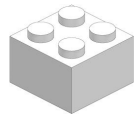
1×4



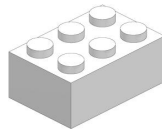
1×6



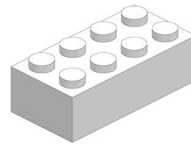
1×8



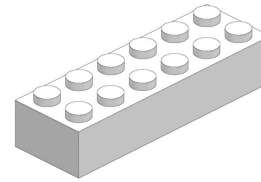
2×2



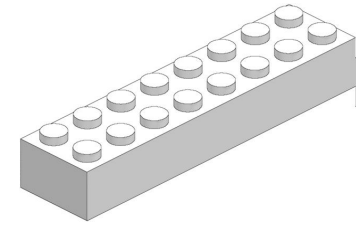
2×3



2×4



2×6



2×8

Basic Greedy Algorithm

1. Consider a single x-z layer at a time
2. Consider a row of voxels at a time
3. Find a group of voxels of the same color
4. If the size odd and > 1 replace with a 1×3 brick
5. Replace with the largest available brick
6. Continue processing the row

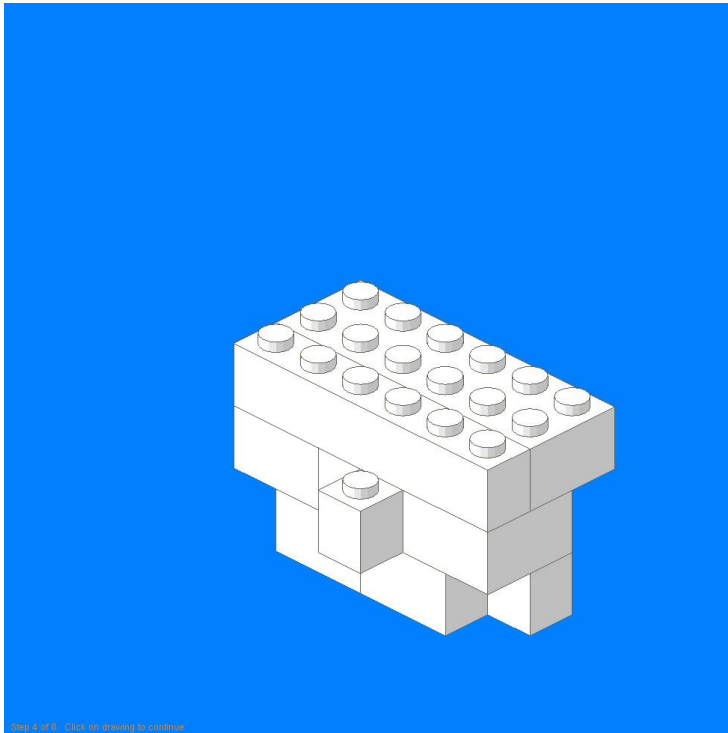
Improvements

- Alternate primary greedy direction between X and Z
- For groups that would be replaced with a 1x1 or 1x2 brick search in the alternate direction
- Use L-shaped corner pieces
- Connect “floating” pieces to model
- Use bricks taller than 1
- Push smaller bricks to the center

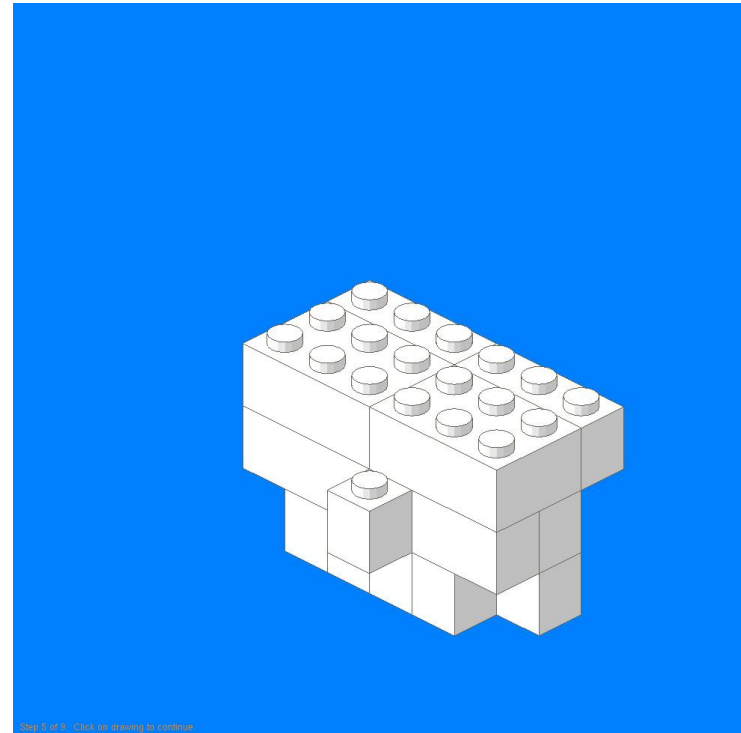
Divide and Conquer Algorithm

1. Consider a single x-z layer at a time
2. Trim any empty rows or columns from the edges
3. If the block of voxels remaining is the size and shape of a brick replace it
4. Otherwise divide into two “equal” halves and repeat for each half

Algorithm Comparison

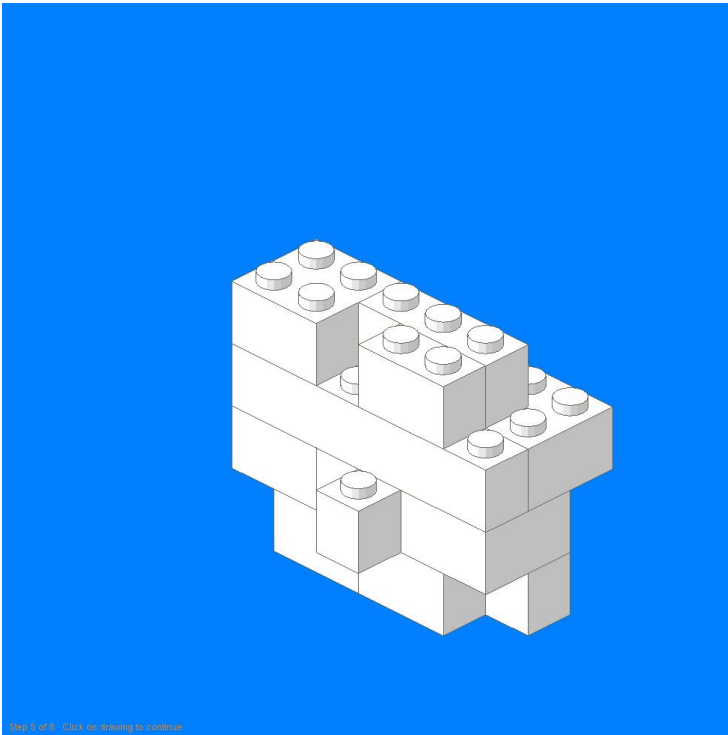


Greedy

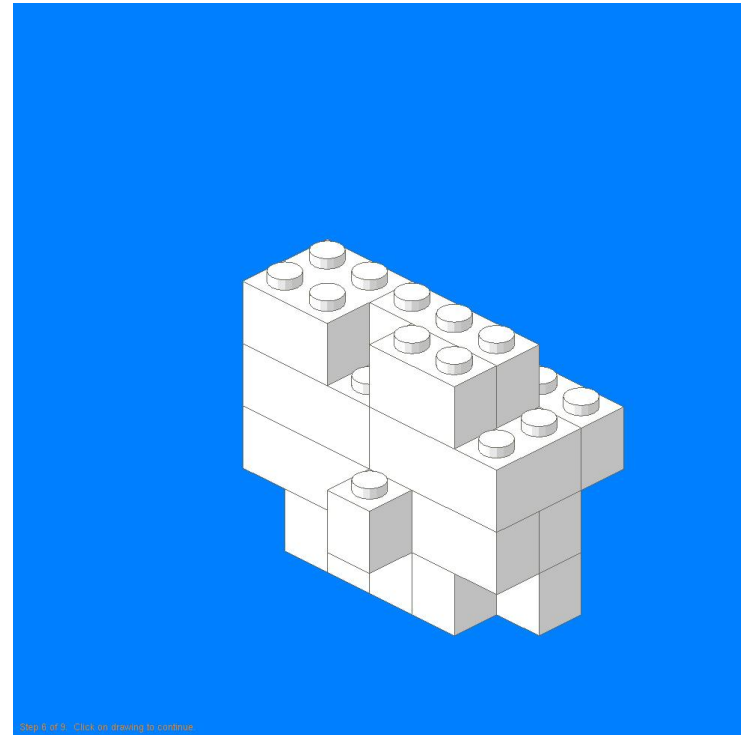


Divide and Conquer

Algorithm Comparison (cont.)

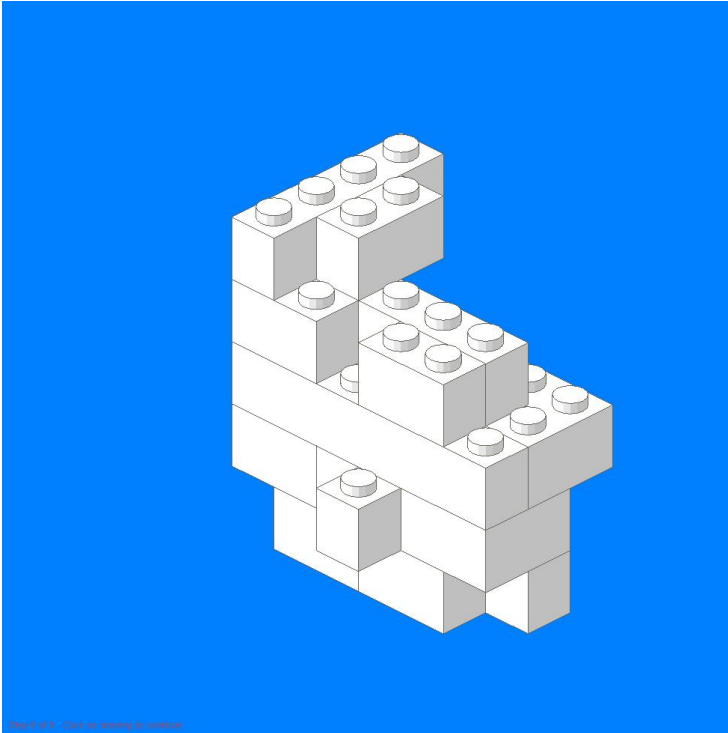


Greedy

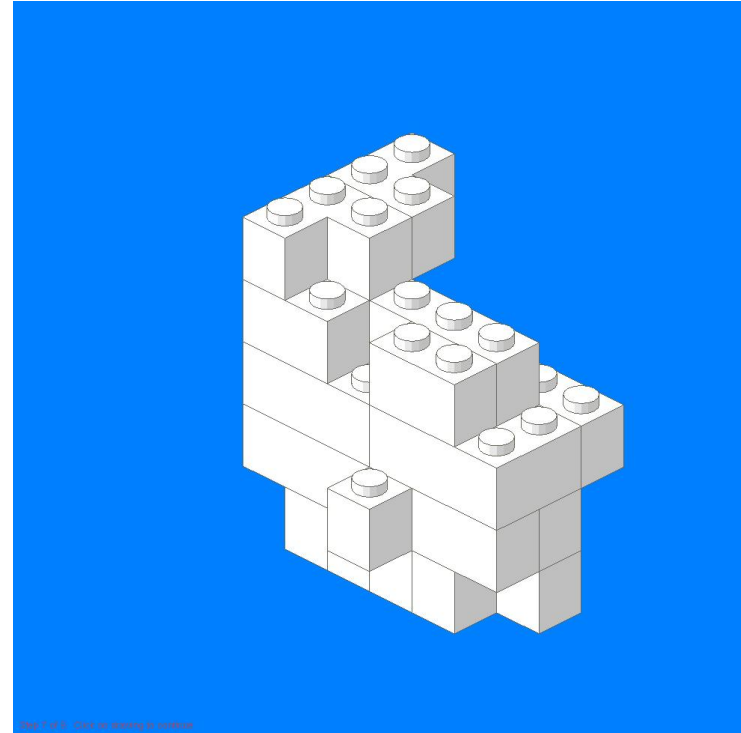


Divide and Conquer

Algorithm Comparison (cont.)

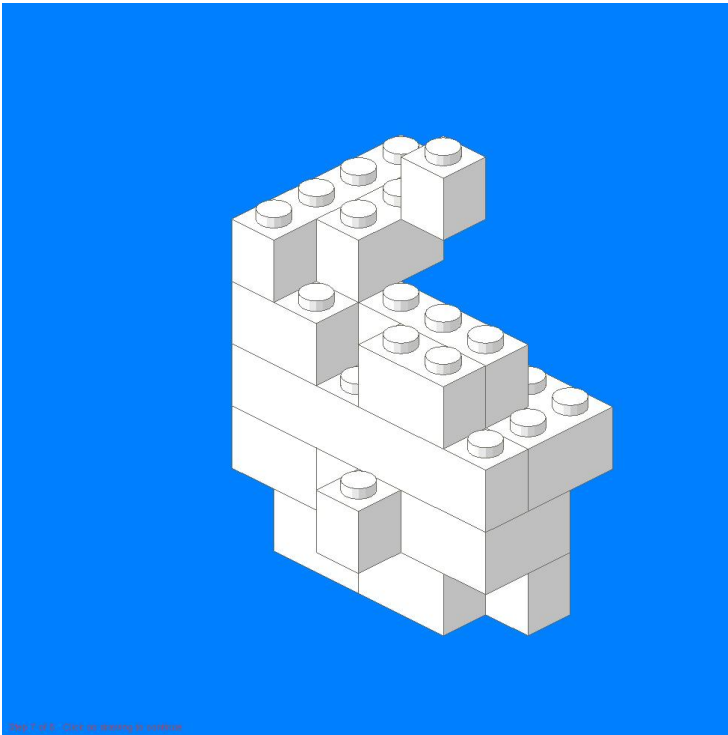


Greedy

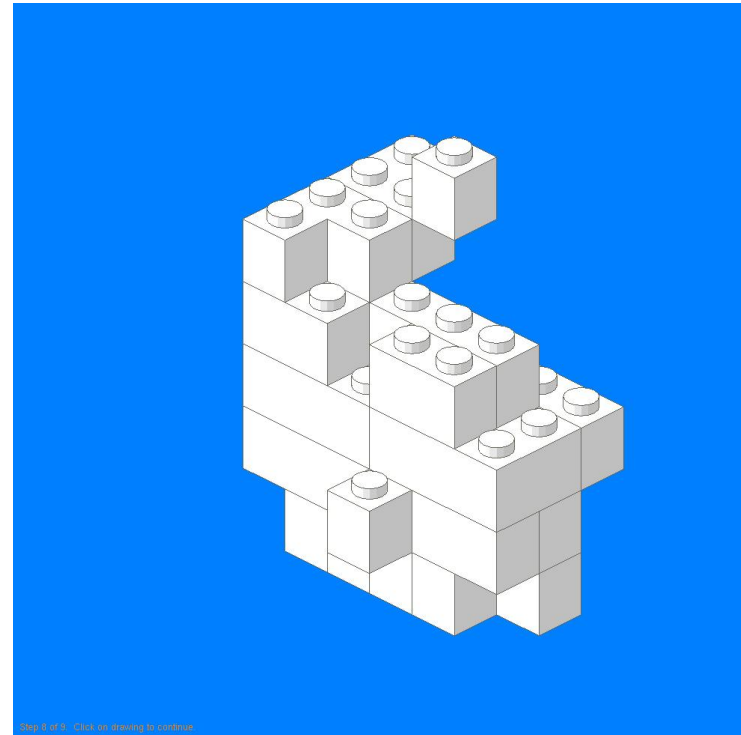


Divide and Conquer

Algorithm Comparison (cont.)



Greedy

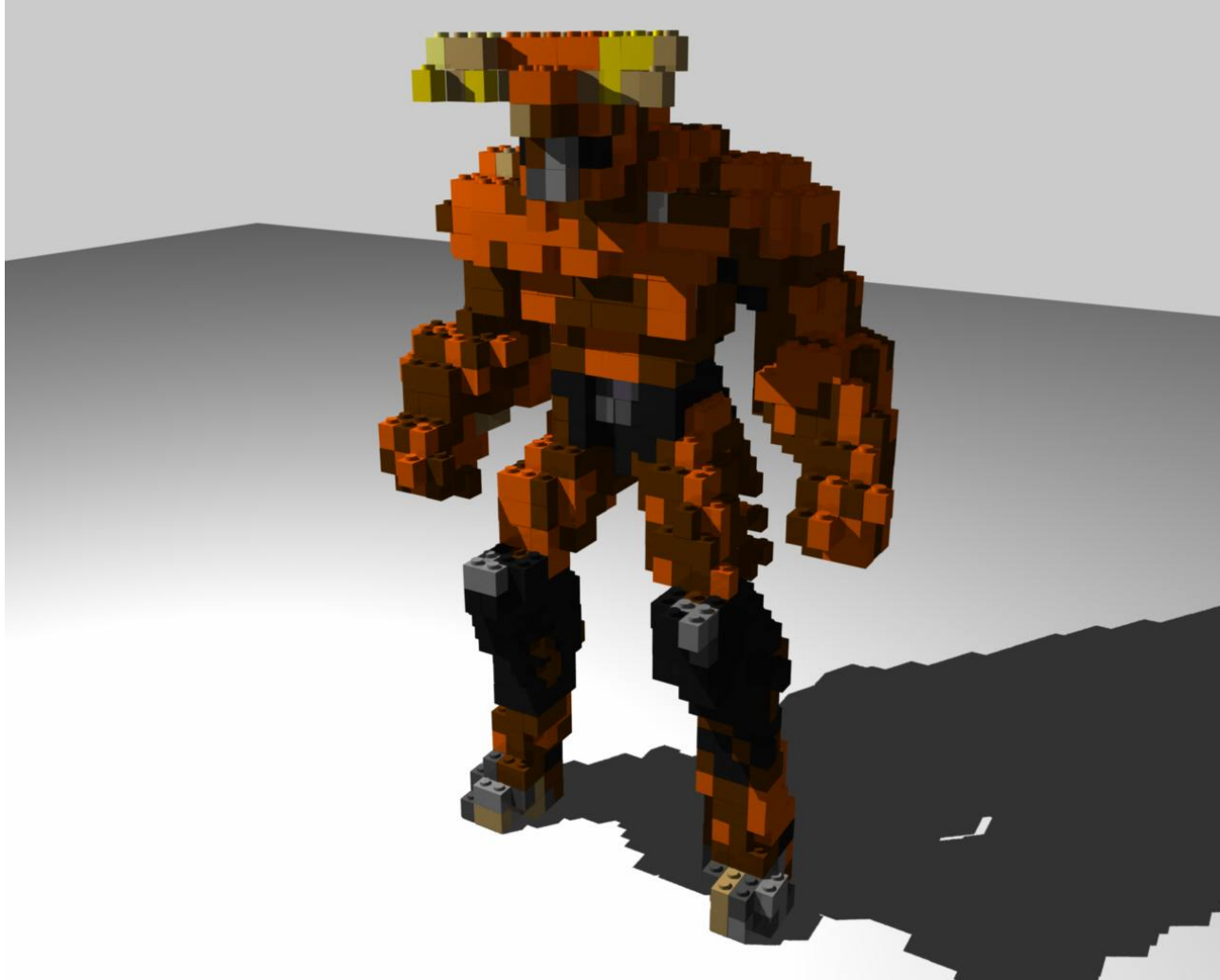


Divide and Conquer

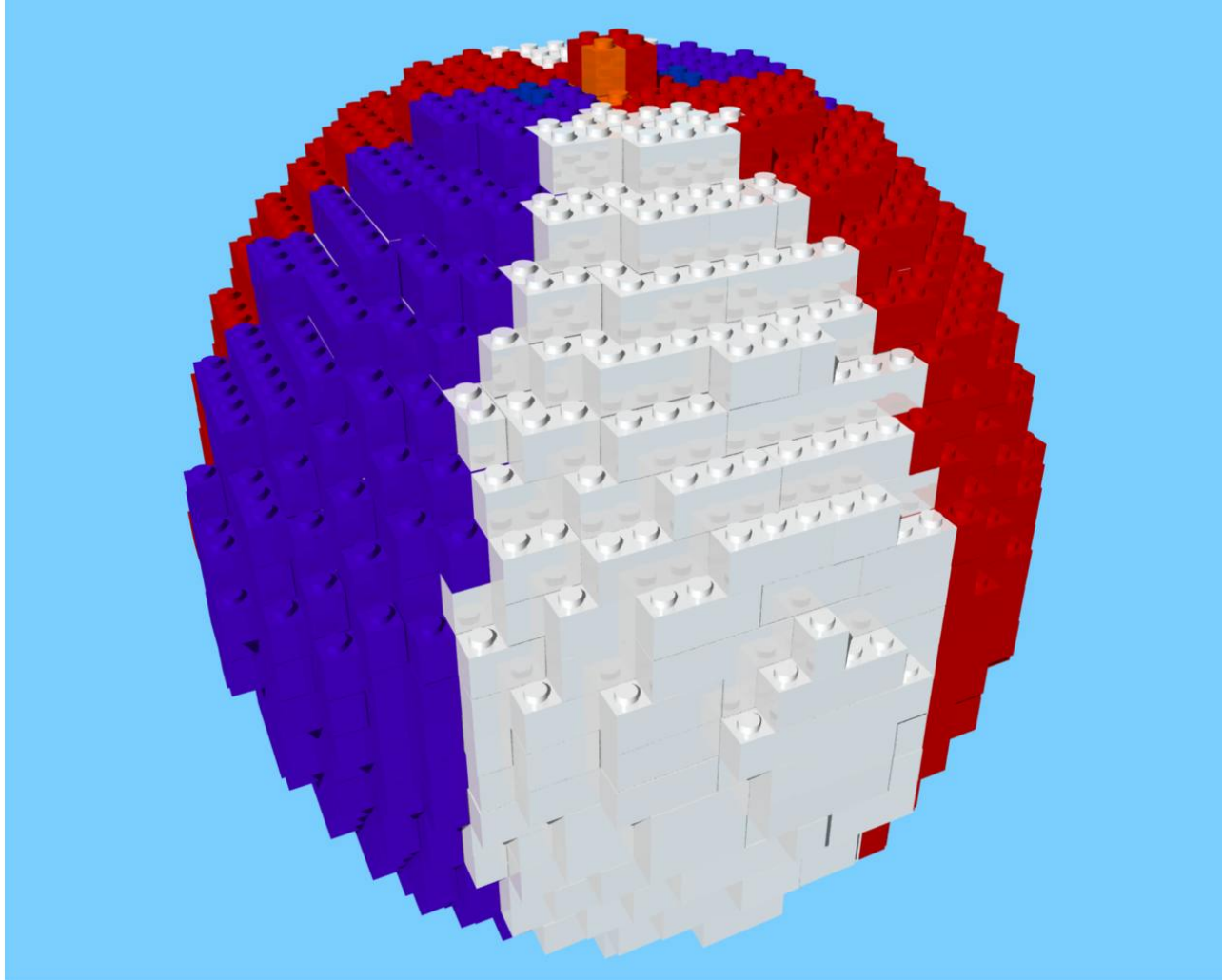


Results

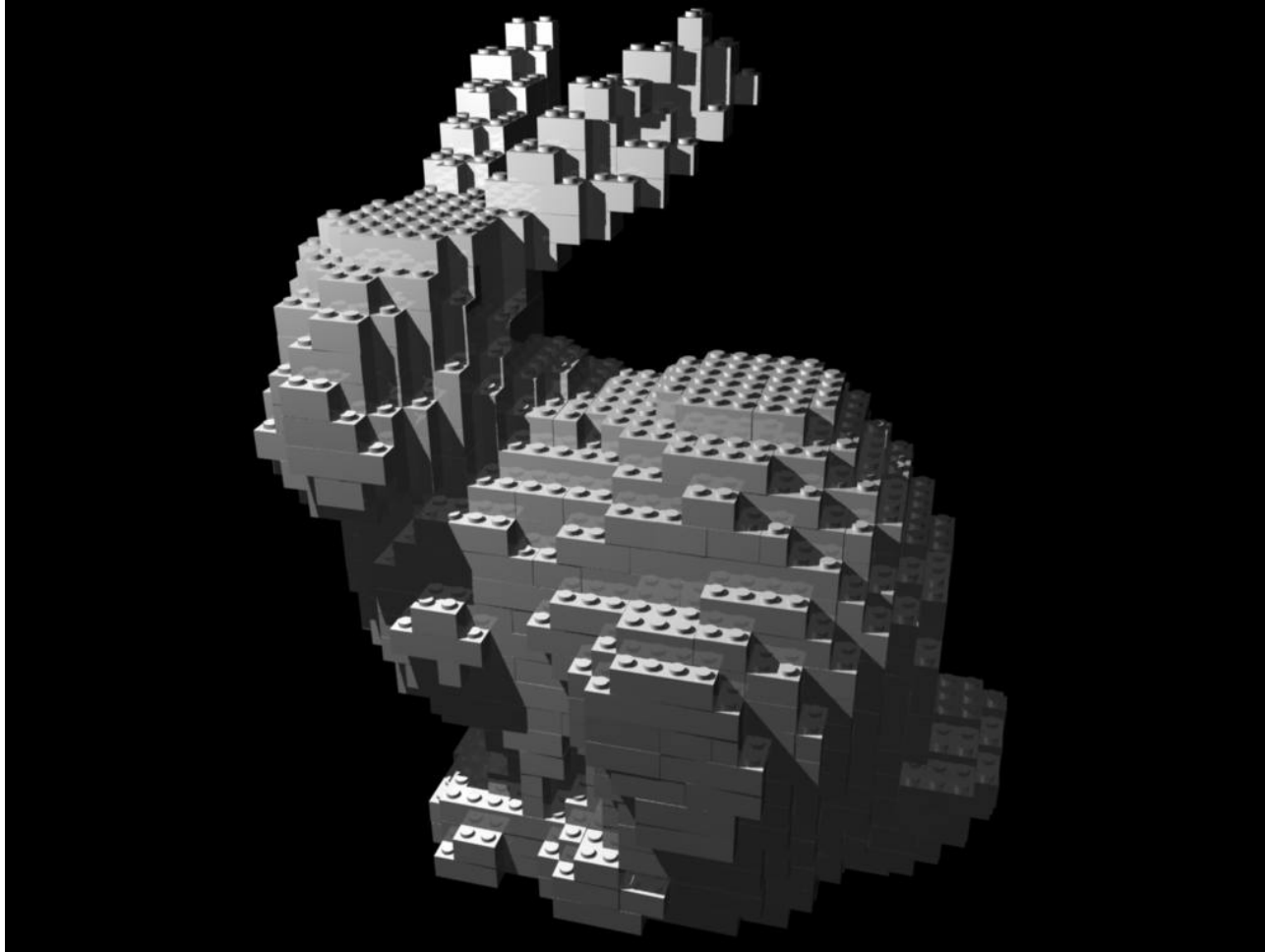
Bauul



Beachball



Bunny



Construction Plans

- Use a program called LPub
- Insert step meta-commands between each X-Z slice
- Example [plans](#)
- Some reworking done during build

References

References

“A Fast Depth-Buffer-Based Voxelization Algorithm”

Evaggelia-Aggeliki Karabassi, Georgios Papaioannou, and Theoharis Theoharis.
ACM Journal of Graphics Tools. 4(4):5-10,
1999.

LDraw website <http://www.ldraw.org>

“Volume Graphics” Arie Kaufman, Daniel Cohen
and Roni Yagel. IEEE Computer. Vol.26, No.
7. July 1993. pg. 51-64.

Media

- All source code publicly available (via CVS)
- uber: <http://graphics.vr.clemson.edu>
- CVS:

<http://jet.vr.clemson.edu/viewcvs/viewcvs.cgi/uber/demos/lego/?cvsroot=UBER>