OpenVCAD: Volumetric Design Tool for Multi-material 3D Printing

Charles Wade & Robert MacCurdy

University of Colorado Boulder

Motivation

Design methods have not kept pace with the advancements in material science and fabrication technologies. In pursuit of creating objects with enhanced mechanical, thermal, and chemical properties, there has been intense research into advanced materials, composites and lattice structures. Likewise, there has been significant advancement in multi-material fabrication through additive manufacturing. New technologies, such as Polyjet and inkjet, enable precise compositional control throughout construction. The combination of these advancements has resulted in *metamaterials* and *digital alloys*. These architected materials vary their composition and structure volumetrically to achieve desirable properties. By optimizing both material composition and geometry, *metamaterials* realize solutions that were previously infeasible with single-material construction. However, there is no design methodology that supports metamaterials.

What is preventing widespread adoption of metamaterials in engineering design?
Answer: Design tools have fallen behind machine capabilities in accommodating metamaterials. Traditional computer aided design (CAD) methods are inadequate for designing objects with multiple materials.

Our work is motivated by the questions: *How can designs efficiently express the volumetric, multi-material composition of objects? How can superior characteristics be designed using metamaterials?*

Approach

In traditional CAD methods, surface representations are ubiquitous because they provide a concise method to characterize object geometry. However, they assume that an object is constructed from a uniformly distributed material. In contrast, multi-material 3D printers expect models to be specified as high-resolution voxels. This representation gives the designer precise control over material distribution, at the cost of enumerating the material value of billions of voxels. To overcome this prohibitive cost, we propose OpenVCAD, an implicit geometry modeling language and compiler for volumetric objects. This method leverages the fact that volumetric data often repeats in patterns and lattices thatvcan be expressed in equations. Using OpenVCAD, engineers can employ a hybrid design approach, combining existing geometric primitives, such as surface representations, with functionally defined material operations. Designs are expressed in code as tree that can be compiled into volumetric data for 3D printing.

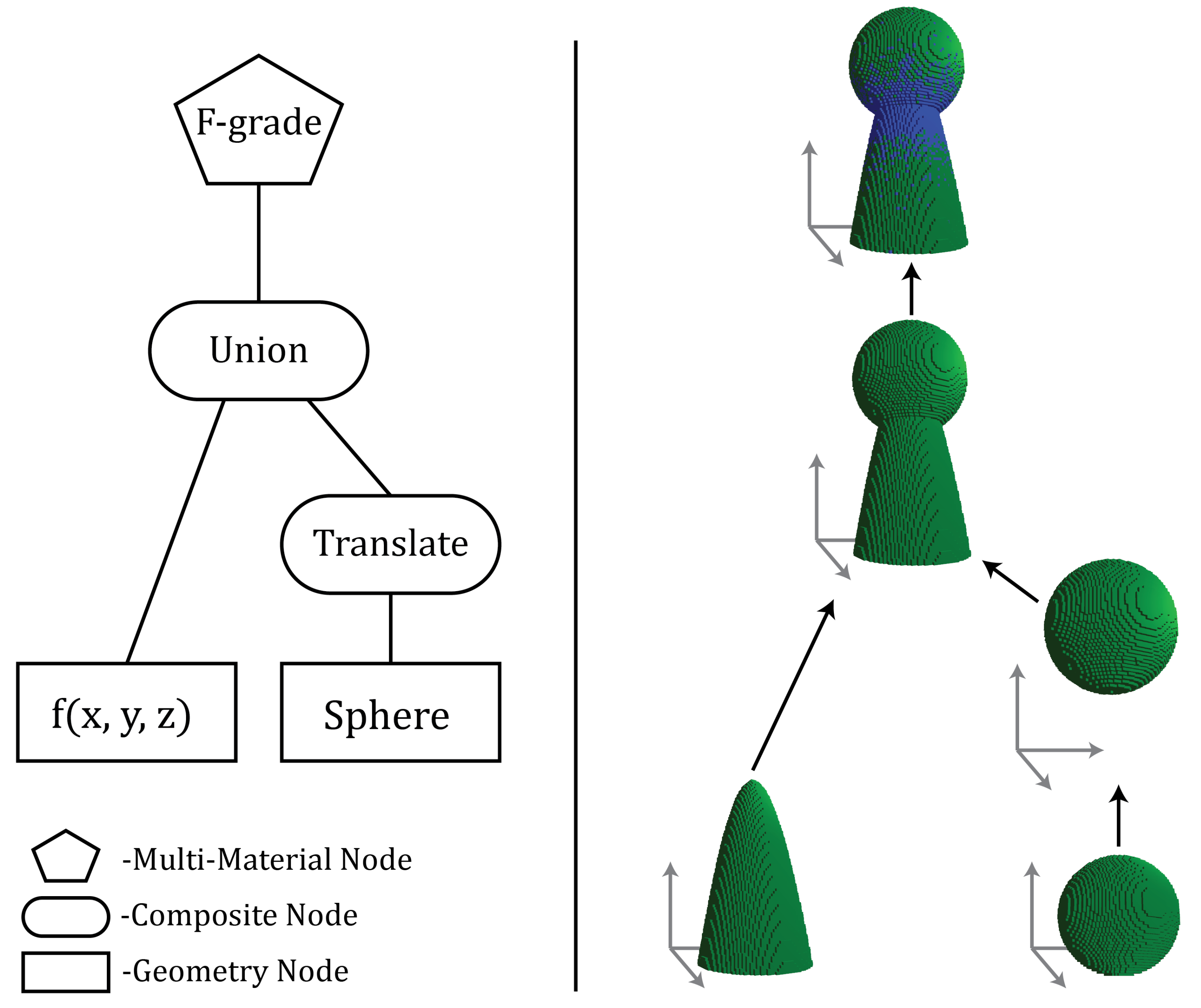

Figure 1. An OpenVCAD tree that defines an object using mixed geometry and materials.

OpenVCAD Modeling Language

OpenVCAD expresses designs using a modeling language that is parsed into a tree. The tree has three type of nodes: geometric, material, and composition. Geometric nodes are the leaves of the tree, and can express predefined shapes, functions, or imported surfaces meshes and voxel data. Material nodes use functions to describe the volumetric composition of child geometry. Composition nodes linearly transform and perform boolean operations on child geometry. The figure below shows the previous tree example in code containing material, composition, and geometric nodes.

```
F-grade(func="normal_dist(z)", materials=[green, blue]) {
    Union() {
      Translate(x=0, y=0, z=5) {
        Sphere(r=3);
      }
      Function_geom(func="parabolic_cone(x,y,z)");
    }
}
```

Figure 2. OpenVCAD modeling code used to define the tree in figure 1

Functional Grading

One way to define volumetric material information is functional grading. The `f_grade()` node employs user-specified, overlapping probability density functions to describe how the material composition of child geometry varies across R^3. When the tree is compiled into voxels for printing, a stochastic sampling process determines the material at a given location. Figure 3 shows the `f_grade()` operation applied to child geometry to create a metamaterial that exhibits enhanced tensile strength by grading a stronger material at the thinner section of the part.

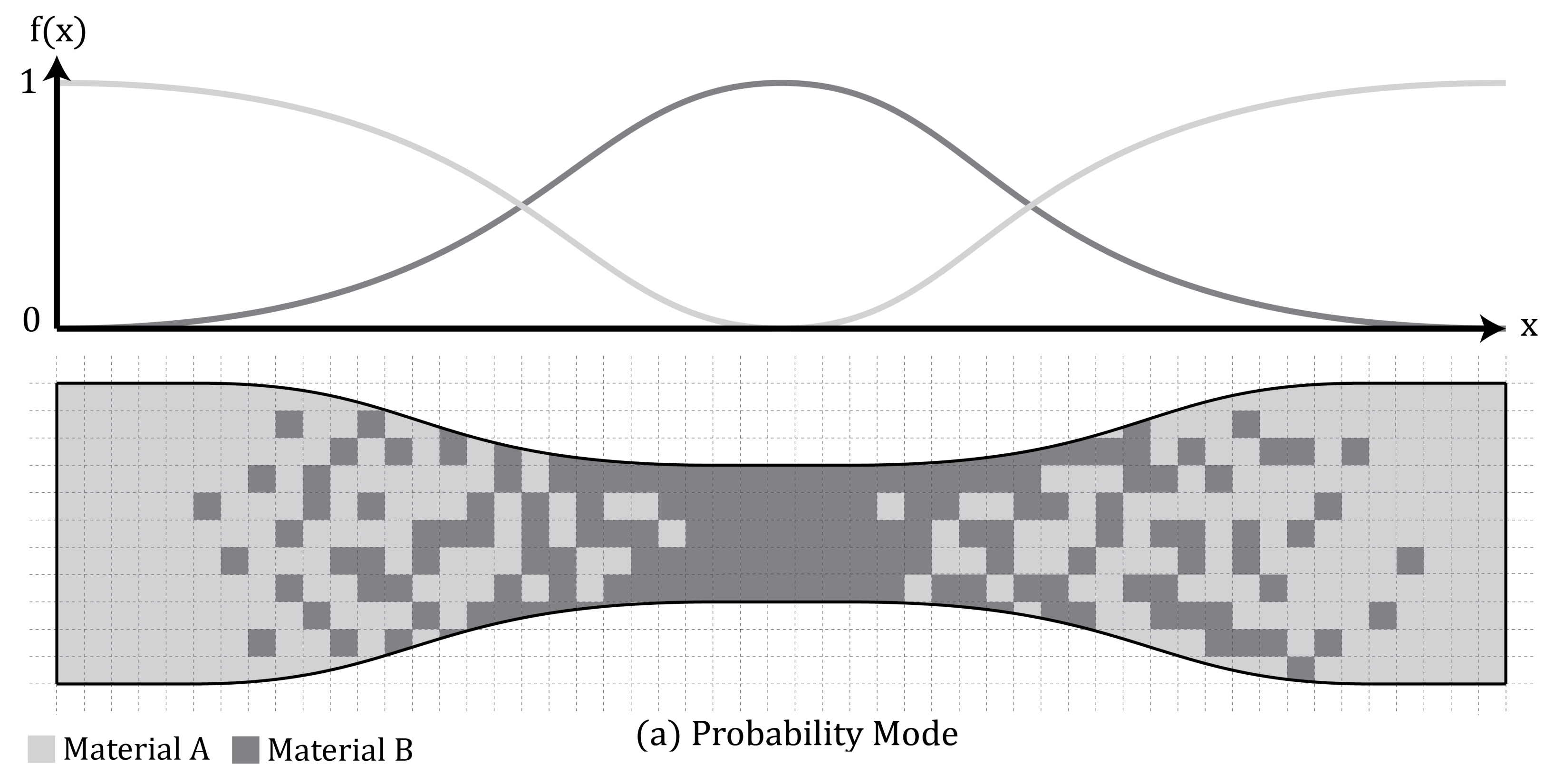

Figure 3. $f\_grade()$ with two overlapping probability density functions applied across an object.

Convolution

OpenVCAD allows users to alter volumetric information through convolution. The `convolve()` node performs discrete 3D convolution using a user-specified kernel. Continuous convolution replaces the center of the kernel with the weighted average of its neighbors. Since there is no relative comparison between two materials, the average in this discrete space can not be used. Instead, the user-specified kernel values are multiplied by the counts of each material to compute a weighted distribution. Then, the center voxel is replaced by the sampling the weighted distribution. The process allows for kernels that can blend (Fig. 4), blur, and sharpen (Fig. 6) material distributions in R^3.

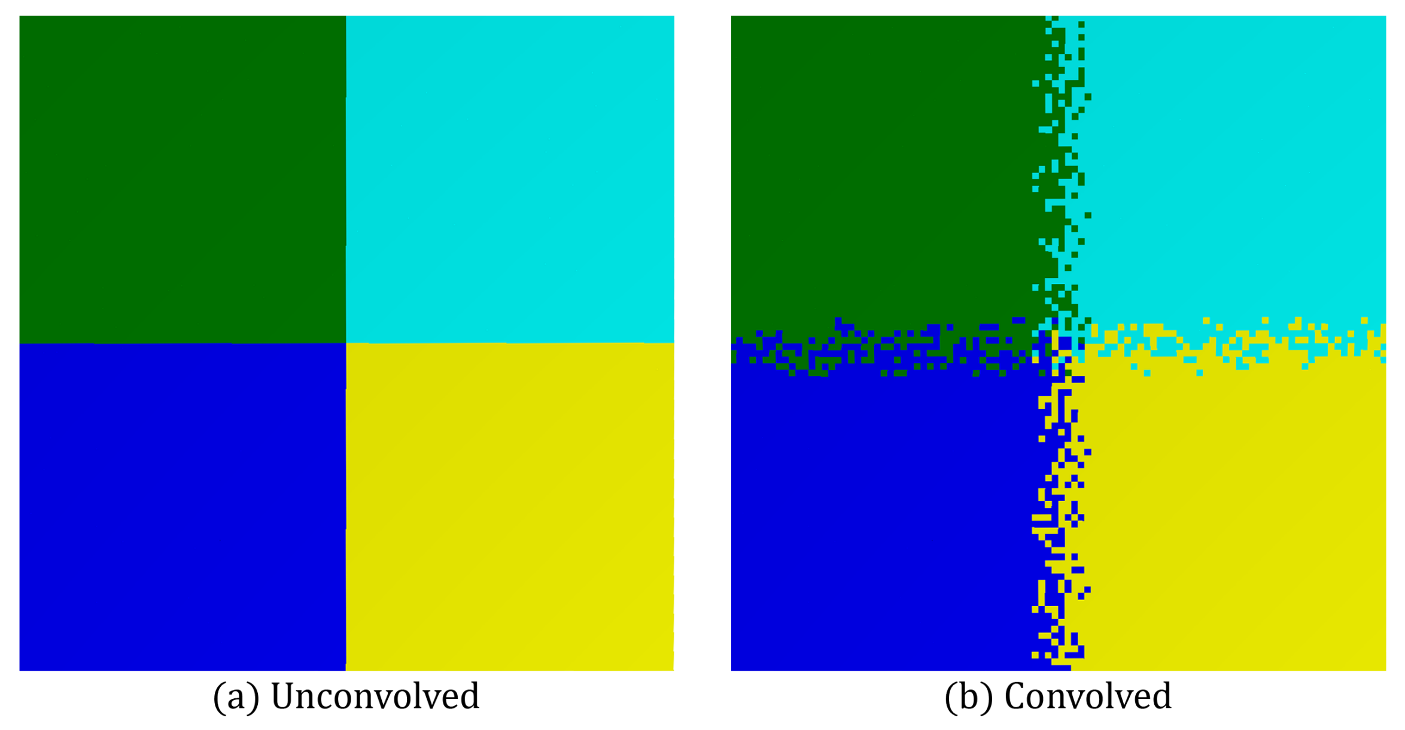

Figure 4. Four squares of different materials are adjacent to each other. A convolution is applied using a blur kernel to intermix materials at the boundaries.

Results

Example 1: Wear Resistant Gear

Figure 5 shows a gear designed and rendered using OpenVCAD. This example demonstrates the `f_grade()` node in a real-world use case. The gear is defined by a surface mesh and a functional grade is applied radially. The green teeth are comprised of a ware-resistant, heavy material that transitions into a lighter, softer material shown in blue. By grading two materials together, we can create a gear that is superior to one comprised of a single material.

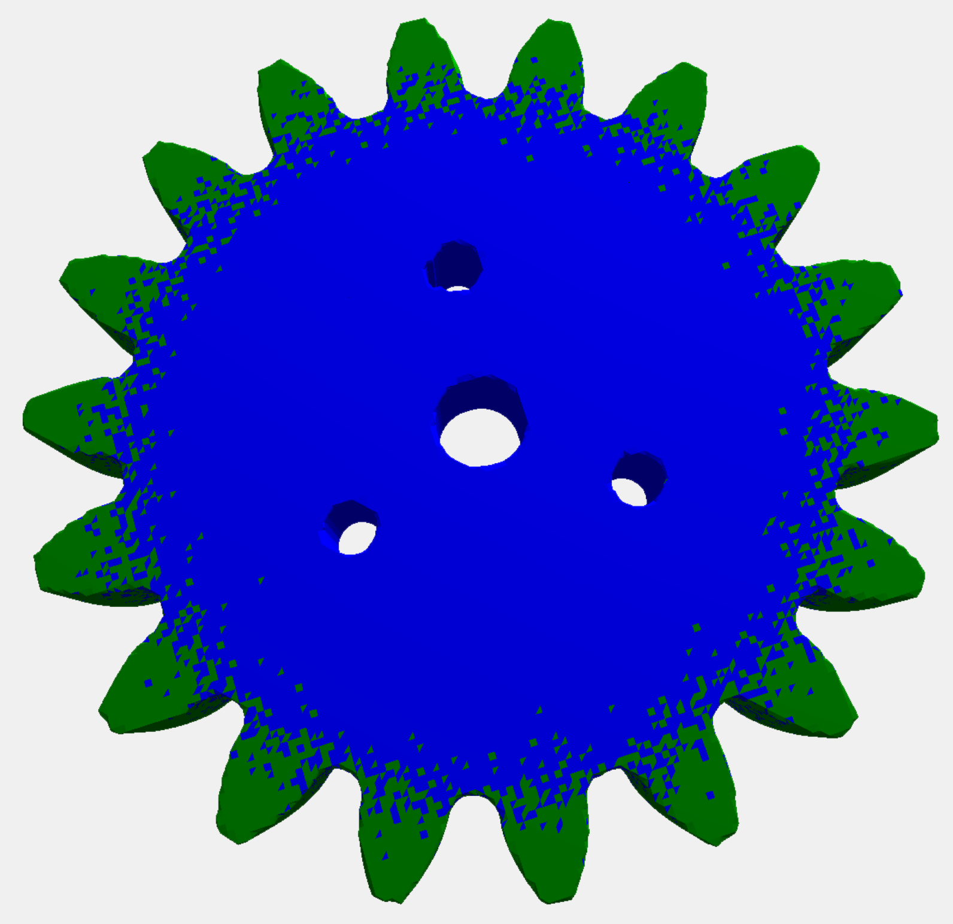

Figure 5. A metamaterial gear designed in OpenVCAD with ware resistant teeth and a lightweight core

Example 2: Medical Imaging Processing

Figure 6(a) shows a brain scan imported and rendered using OpenVCAD. When 3D printed, the pink hot spots will be obscured by the blue regions throughout the model. To reduce this noise, the `convolve()` node was applied with a sharpening kernel. The result in 6(b) shows a cleaned cross-section that allows for easier visualization.

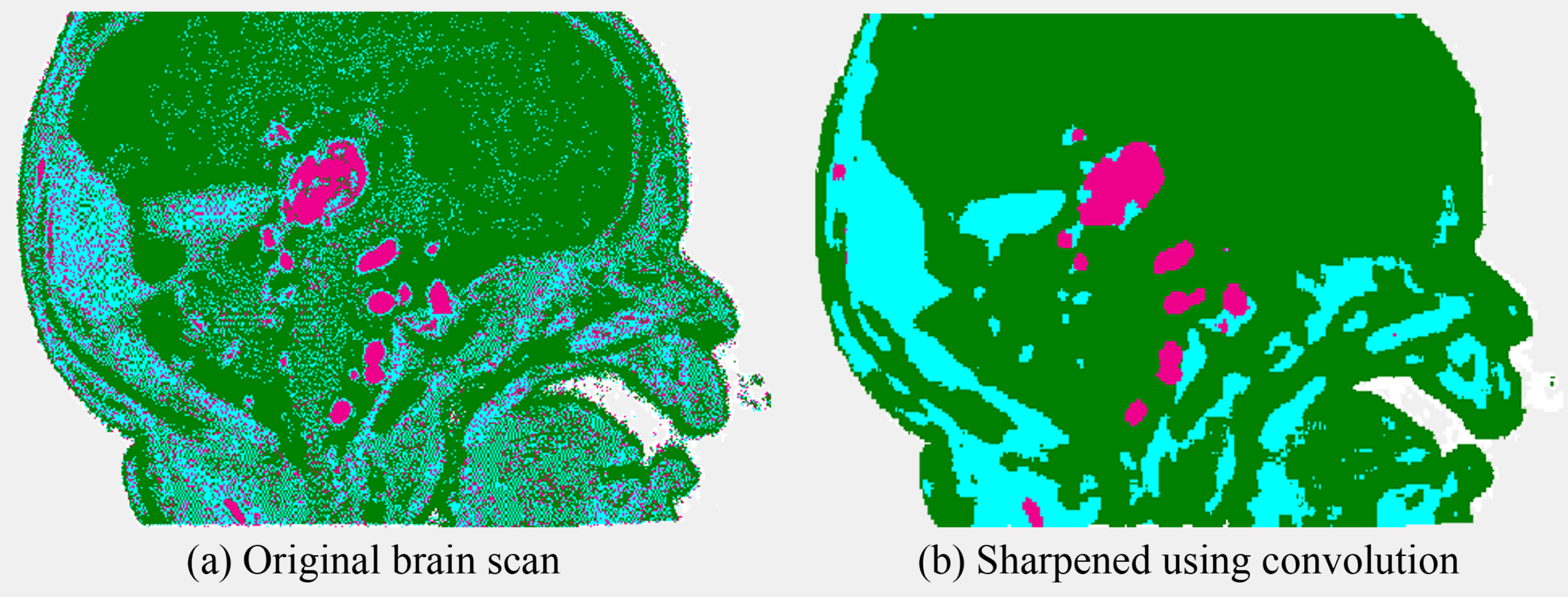

Figure 6. An input brain scan (a) is cleaned to yield (b) using the `convolve()` node and a sharpening kernel.