Spec2Fab: A Reducer-Tuner Model for Translating Specifications to 3D Prints

Desai Chen David I.W. Levin Piotr Didyk Pitchaya Sitthi-Amorn Wojciech Matusik

MIT CSAIL

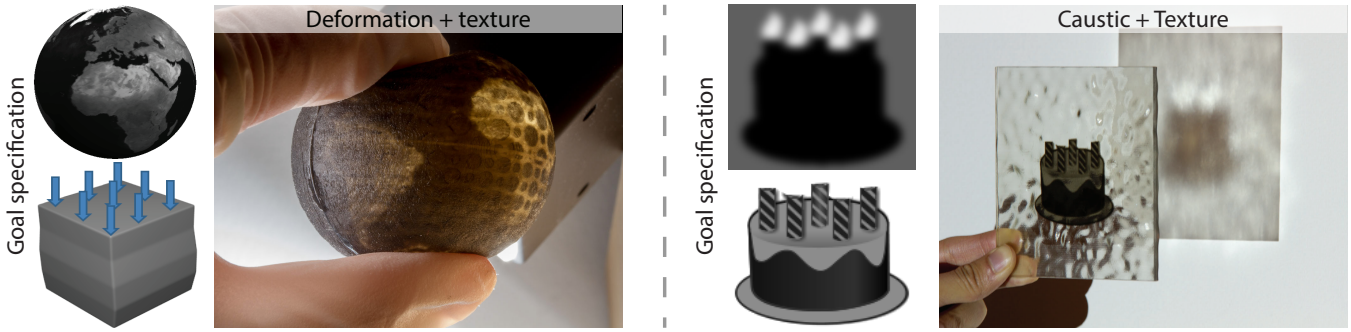

Figure 1: *3D-printed objects with various effects designed using our* reducer-tuner *model. Our generalized approach to fabrication enables an easy and intuitive design of objects with different material properties. On the left: a miniature of Earth with a prescribed deformation behavior. On the right: an optimized surface producing a caustic image under proper illumination as well as casting a shadow of a previously designed shape. Insets visualize an input to our system.*

Abstract

Multi-material 3D printing allows objects to be composed of complex, heterogeneous arrangements of materials. It is often more natural to define a functional goal than to define the material composition of an object. Translating these functional requirements to fabricable 3D prints is still an open research problem. Recently, several specific instances of this problem have been explored (e.g., appearance or elastic deformation), but they exist as isolated, monolithic algorithms. In this paper, we propose an abstraction mechanism that simplifies the design, development, implementation, and reuse of these algorithms. Our solution relies on two new data structures: a *reducer tree* that efficiently parameterizes the space of material assignments and a *tuner network* that describes the optimization process used to compute material arrangement. We provide an application programming interface for specifying the desired object and for defining parameters for the *reducer tree* and *tuner network*. We illustrate the utility of our framework by implementing several fabrication algorithms as well as demonstrating the manufactured results.

CR Categories: I.3.8 [Computer Graphics]: Applications;

Keywords: 3D printing, goal-based material design, fabrication

Links: DL PDF WEB DATA CODE

1 Introduction

3D printing receives a lot of attention as it aims to democratize fabrication. The ever expanding range of printing materials allows for fabrication of complex objects with spatially varying appearance, optical characteristics, and mechanical properties. One of the most important unsolved problems in this area is how to compute an object's material composition from a functional or behavioral description. We call this process *specification to fabrication translation* (Spec2Fab). The goal of this work is to provide a convenient abstraction for specifying such translators. This is necessary to move past the current direct specification model of 3D printing.

Today, 3D printing of an object requires a material be directly specified for each voxel inside the object volume. This approach is fraught with difficulties. First, 3D printable models become specific to a single printer type, i.e., the models are built from materials provided by a given printer. Consider the inconvenience that would result from word processing documents being compatible with specific 2D printers. Second, working directly with printing materials rather than material properties is extremely challenging for users. Imagine the difficulty in finding the right combination of printing materials that would provide a specific color, stiffness, or refractive index.

Our work is motivated by the recent research efforts in the computer graphics community to create specific instances of the translation process, for example, subsurface scattering [Hašan et al. 2010; Dong et al. 2010] or deformation properties [Bickel et al. 2010]. However, each of these instances is a custom, monolithic solution which is difficult to extend, combine, or modify. Our main insight is that all these process instances share a similar structure. First, they rely on the ability to accurately simulate the physical properties of an object given its geometry and material assignment. They use this simulation within an optimization framework to search the space of all possible material assignments in order to find the one that best reproduces the desired properties. Due to the combinatorial nature of the search space the naive optimization approach is not tractable. For example, when the printing volume has N voxels and each of these voxels can be assigned to one of M base materials, the search space has N^M dimensions. To overcome this problem, the search space is reduced to a lower-dimensional space using a

reduction model. The goal of the reduction step is to aggressively shrink the search space in a domain-specific manner such that it still contains good approximations to the optimal solution. This search space reduction combined with the right choice of the optimization algorithm delivers a computationally tractable approximation.

The reduction-optimization structure suggests that it is possible to provide a more general abstraction mechanism for translating 3D models to printer and material-specific representations. In this paper we take the first step in achieving this goal. Our solution relies on two novel data structures which are designed to aid the fabrication process. The *reducer tree* is a tree-based data structure that allows us to parameterize the space of material assignments. The *tuner network* is a data structure for specifying the optimization process. Our solution also provides an API for specifying the desired object, setting up the simulation, and defining parameters for the *reducer tree* and *tuner network*. In general, our framework simplifies the construction of new computational fabrication algorithms. More specifically, different components of the process can be easily replaced and other components easily reused. Various optimization strategies can also be explored with lower implementation burden. In order to show these advantages, we illustrate how existing computational design processes fit into this framework and how they can be combined. We demonstrate the results of these algorithms on a variety of different examples fabricated using 3D printers.

2 Related Work

Our new data structures draw ideas from previous work in rendering and optimization. The *reducer tree* is inspired by Cook's shade trees [Cook 1984] and their modern implementation in current rendering systems (e.g., Maya, RenderMan, etc.). Using these approaches, complex effects can be achieved by combining a set of basic shading blocks. The *reducer tree* also uses a tree data structure that combines a set of primitives to compute a material assignment for each point inside of an object volume and describes a spatial-partition of the object volume. While there are many space-partitioning data structures (BSPs, Quad-trees, KD-trees, etc.), they are difficult to specify by hand and they are typically tied to the object geometry. Our *reducer tree* is intuitive to construct but sufficiently general for representing material distributions. In addition, it is specified in an object-independent manner – the same *reducer tree* can be reused for processing different objects. Our second, new data structure which is responsible for optimizing material assignment is the *tuner network*. It is inspired by probabilistic graphical models [Jordan et al. 1999] that have been popular for representing variable dependence in multi-variate optimization problems. These problems can be parallelized [Low et al. 2010] according to the associated graph structure.

Our work is also inspired by the many instances of specification to fabrication translation pipelines that have been proposed recently. These pipelines span a wide range of functional goals (e.g., mechanical, optical, appearance). They drive the simulation, optimization, and reductions chosen for each method. One example of such processes is recent work on optimizing material composition to achieve prescribed deformation behavior. Bickel and colleagues [2010] have designed a system for manufacturing multi-layer composites with a given elastic behavior. Skouras *et al*. [2012] provide design and construction of balloons with a prescribed shape while inflated. Furthermore, material optimization has been employed to create mechanical clones of human faces [Bickel et al. 2012].

Optimizing and manufacturing objects with desired appearance and optical characteristics has also been explored. Weyrich *et al*. [2009] compute surface micro-geometry that yields a desired BRDF. Similar approaches have been proposed [Finckh et al. 2010; Papas et al. 2011] to produce refractive surfaces that form user-defined caustics. These methods have been extended to optically decrypt hidden images [Papas et al. 2012]. Complementary work examines fabricating surfaces with spatially varying reflectance [Matusik et al. 2009; Malzbender et al. 2012] and diffuse shading [Alexa and Matusik 2010]. Another set of approaches uses optimization to compute shadow casting surfaces and volumes [Mitra and Pauly 2009; Bermano et al. 2012; Baran et al. 2012] that reproduce a given set of input images. Finally, optimization-based approaches have also been employed to control the subsurface scattering of 3D printed multi-layered models [Dong et al. 2010; Hašan et al. 2010]. We seek to exploit the common form of the above works in order to generalize them.

There have been studies of material assignment representations in other fields, primarily in mechanical engineering. Here we only list a few representative works. Kumar *et al*. [1999] describe material composition by dividing a volume into sub-volumes. They perform material interpolation using local, sub-volume coordinate systems. Jackson [2000] explores several mesh data structures for spatial sub-division. Kou and Tan [2007] give a comprehensive review of spatial partition schemes and material interpolation functions. Kou *et al*. [2012] use a hierarchy of procedures to define material composition with a small number of design parameters. They run particle swarm optimization [Kennedy and Eberhart 1995] on the design parameters to minimize thermal stress of an object. Their work focused on smoothly varying materials. There is no discussion of high-frequency discrete material assignment for capturing details. They are limited to global optimization algorithms such as Particle Swarm Optimization because the dependency structure between the design variables is not modeled. In computer graphics, procedural descriptions for material assignment have also been studied. Cutler *et al*. [2002] describe a scripting language for specifying layered solid models and describing material composition. Vidimče *et al*. [2013] provide a fabrication language and a programmable pipeline for specifying material composition directly and precisely throughout a volume. In contrast, Spec2Fab can be used to design algorithms that only require object properties rather than a precise description of material assignment.

In this work we focus on constructing a common model for encompassing the entire Spec2Fab problem. We are motivated by the similarities present in state-of-the-art algorithms. Table 1 shows the coordinate reduction methods as well as the optimization schemes used in all these prior approaches. Our key observation is that previous works share a common methodology and a common set reduced coordinate components. These components are then combined with (often off-the-shelf) optimization techniques to form new material assignment algorithms. This suggests that one could find a small set of components that can be tied together to synthesize advanced methods. In this paper we attempt to identify these components as well as a suitable model for their interaction.

3 Design Goals

The design of our general translational framework is guided by the following principles:

- **Modularity:** Spec2Fab translators are complicated both algorithmically and from a software engineering point of view. To combat this, any proposed framework must break the problem into a manageable number of small, reusable building blocks.
- **Extensibility:** Developers must be able to add their own building blocks to the system. This allows the system to grow in conjunction with the capabilities of newer 3D printers.

Paper	Goal	Reduced Coordinates	Optimization
[Finckh et al. 2010]	Optical (Caustics)	B-Spline Surface	Stochastic Approximation
[Papas et al. 2011]	Optical (Caustics)	Piecewise Constant Tiles	Simulated Annealing
[Alexa and Matusik 2010]	Optical (Relief)	Height Field	Gradient Descent
[Weyrich et al. 2009]	Optical (Reflectance)	Piecewise Constant Tiles	Simulated Annealing
[Papas et al. 2012]	Optical (Refraction)	Piecewise Constant Tiles	Simulated Annealing
[Mitra and Pauly 2009]	Optical (Shadows)	Voxel Grid	Custom Discrete
[Baran et al. 2012]	Optical (Shadows)	Layered Materials	Quadratic Program
[Bermano et al. 2012]	Optical (Shadows)	Height Field	Custom/Simulated Annealing
[Dong et al. 2010]	Optical (Subsurface)	Layered Materials	Conjugate Gradient
[Hašan et al. 2010]	Optical (Subsurface)	Layered Materials	Branch and Bound
[Bickel et al. 2010]	Mechanical (force)	Layered Materials	Branch and Bound
[Bickel et al. 2012]	Mechanical (shape)	Height Field	Newton-Raphson
[Skouras et al. 2012]	Mechanical (shape)	Triangle Mesh	Augmented Lagrangian Method

Table 1: *The goal type, reduction type and optimization used by prior computational fabrication approaches.*

- **Device Independence:** Spec2Fab translators should be device independent. They should be easily adaptable to different types of 3D printers.
- **Input Geometry Independence:** Spec2Fab translators should be geometry independent. For example, a process for applying a texture to a 3D printed object should work for **any** object.

4 Overview

We aim to separate the Spec2Fab process into two phases, the process configuration phase and the process use phase. The process configuration phase is typically done once by a skilled developer who constructs a Spec2Fab translator. The process use phase is typically performed multiple times by an end-user who is only required to provide an object specification (e.g., object geometry and deformation properties) and a target device.

The process configuration phase produces a Spec2Fab translator which will assign a desired volumetric material distribution to a user supplied input geometry given a user specified goal. We describe this phase using two new data structures, the *reducer tree* and *tuner network*. The *reducer tree* parameterizes a volumetric material assignment using a small set of *geometry* and *material nodes* while the *tuner network* is used to describe an optimization process as a connected set of *tuner* objects. Both *nodes* and *tuners* can be easily recombined and reused thus making our framework highly modular. Furthermore, *nodes* and *tuners* define abstract interfaces and thus developers can easily add new types of each. This makes our framework extensible.

Reducers and *tuners* are chosen to be independent of printer capabilities. Instead we account for printer type by altering the available materials that the translator may assign to the input geometry. This is important since it grants the *reducer-tuner* model device independence. Finally, the *geometry nodes* of the *reducer tree* are designed to function irrespective of input shape. Since the Spec2Fab translators produced by our algorithm use a composition of these node types, they are geometry independent.

In the following sections we will describe the *reducer tree* and the *tuner network*, their constituent components and the mechanisms by which they interact.

5 Data Structures

In this section, we describe the structure of the *reducer tree* as well as all types of *reducer nodes* implemented in our framework. We also show the structure of individual *tuners* and how they can be arranged into the *tuner network*.

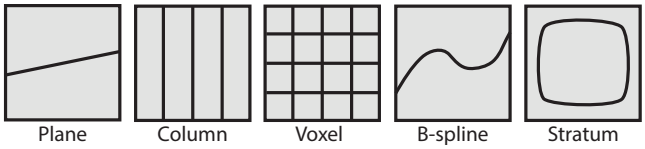

Figure 2: *2D representations of the* geometry nodes *used in our* reducer tree.

5.1 The Reducer Tree

Estimating material assignments at output device resolution is computationally intractable. Therefore, material assignment has to be computed using a reduced representation. We specify this representation using a *reducer tree*. This structure is conceptually similar to those used in programmable shading systems (such as Cook's shade trees [1984] or Maya Shader Networks). These systems are primarily concerned with assigning known materials and textures to an object's surface whereas we are seeking an optimal volumetric assignment from a defined set of materials. In order to accomplish this task, we build a tree-based data structure which contains the entire object volume as its root node. We define two classes of *reducer nodes*: *geometry nodes* and *material nodes*.

Geometry Nodes: A *geometry node* takes a volumetric region as input and produces a partition of this region into smaller subregions. These can be attached to the object geometry or they can be defined in a global coordinate system. To demonstrate the flexibility of the *reducer tree*, we define a small yet powerful set of partitioning nodes which are described as follows (Figure 2):

- Plane Node – partitions the space into two half-spaces,
- Column Node – takes as input the number of columns and accordingly partitions the space,
- Voxel Node – takes as input voxel size and uniformly partitions the space,
- B-spline Node – partitions a volume into two regions cut by a B-spline,
- Stratum Node – takes a single positive distance parameter as an input and partitions the volume into two regions divided by the iso-distance surface.

Material Nodes: The *leaf nodes* of the *reducer tree* are the material parameterization nodes. These contain a material assignment

function $\lambda(x)$ that maps spatial coordinates to materials. We allow our *material nodes* to assign a void material to regions of the volume. Hence, surface displacements and material assignments can be treated in a unified fashion. While *material node* can be extended to implement arbitrary material assignment functions (such as Functionally Graded Materials), all the following results require only a single type of *material node*, *layered node*, which assigns k material layers of varying thickness to a geometry partition. We can even assign a constant material to a region using a *layered node* with a single layer.

Geometry nodes and *material nodes* can be connected into a tree structure that describes a material assignment function throughout the input geometry. The essential property of this data structure is that it naturally adapts to the input geometry. This key feature allows it to be reused for different shapes. Figure 3 shows two examples of *reducer trees* and the resulting geometric distributions of materials. Since the tree describes a nested set of partitions, we can efficiently perform material queries by passing a query point through the tree until it arrives at a parameterization node. We note that the *reducer tree* does not inherently enforce material continuity between disjoint regions of the object; however, for material assignment problems, this is neither required nor desired.

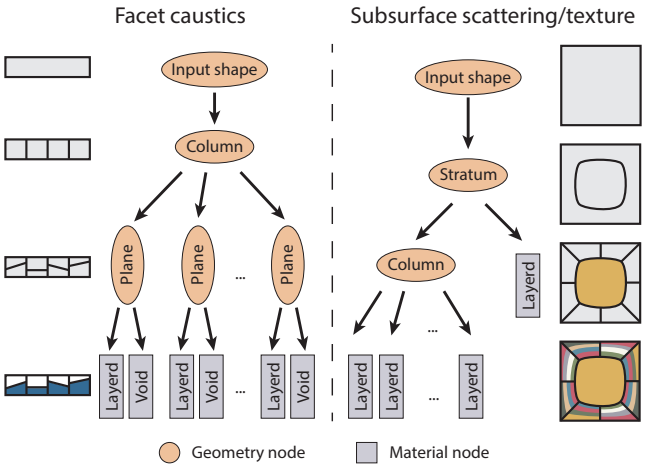

Figure 3: *Two examples of our* reducer tree. *Left: in order to create an object producing a caustic, the input shape is first divided into columns. Next, each column is sliced by a plane. By assigning materials to lower parts of the columns, the microfacet surface is created. Right: a textured object with subsurface scattering properties is produced by first dividing the input shape using a* stratum node. *Then, the outer layer is sliced into columns, which create a texture once materials are assigned. The inner part of the object obtains a material with given subsurface scattering properties.*

5.2 Tuners and Tuning the Reducer Tree

We define *tuners* as an abstract interface between a *reducer node* and an optimization algorithm. A *reducer node* has parameters that control its space partitioning and material composition, e.g., *stratum node* has a parameter which determines its thickness, *column node* has parameters that control the width of columns. A *tuner* is responsible for tuning these parameters to achieve a specified goal. A *tuner* is comprised of an optimization scheme, an error metric, a simulator and a goal (Figure 4). In order to create an optimal material assignment, *tuner nodes* must be attached to the *reducer tree*. The developer links each *tuner* to a *reducer node* in the *reducer tree*. Each *tuner* then traverses the *reducer subtree* (rooted at this *reducer node*) and constructs a parameter list by querying each visited *reducer node*. The developer can label parameters as fixed

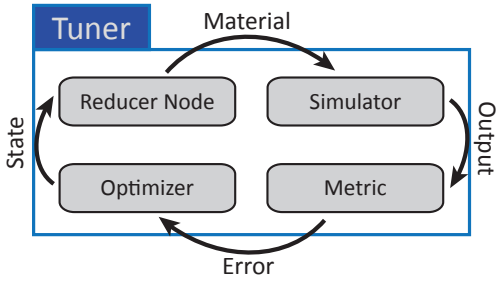

Figure 4: *A diagram of the* tuner *workflow. Arrows indicate flow and type of information passed between the individual* tuner *components.*

or free. *Tuner nodes* contain specific optimization routines (e.g., a quadratic program solver). The *tuner node* then uses this routine to optimize its associated free parameters. During execution, additional information (e.g., parameters, errors, etc.) from neighboring Tuner Nodes can be obtained via the *tuner network* (Section 5.3). Tuner execution can be scheduled (again by the programmer) allowing both serial and parallel processing.

Figure 5 (left) shows a simple example in which two *tuners* are attached to sibling nodes in a *reducer tree*. The first *tuner* is responsible for *layered* and *void material nodes* while the second is responsible for several *layered material nodes*.

5.3 Tuner Network

For many fabrication problems, *tuners* should not act in isolation. For instance, an optimal material assignment at a given point depends on the material assignment in the neighborhood. Tuning the free parameters without taking into account this dependency usually yields suboptimal results. Therefore, we allow the *tuners* to share information according to a user-specified graph structure. This allows our framework to interface with a wide range of existing graph-based optimization and inference algorithms. For example, Figure 5 (right) shows one of the *tuner networks* used in our experiments. In this network, each node is connected to its four neighbors. In Figure 11, *tuners* enclosed in dashed boxes are connected in this form. *Tuner networks* do not always exhibit this regular connectivity. For instance, *tuners* can be completely unconnected (Figure 5 (left)) or be organized into groups which feature intra- but not inter-group connections.

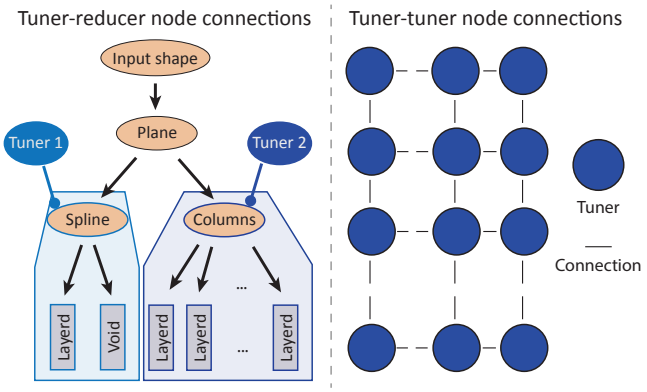

Figure 5: *Left: Two* tuners *attached to a* reducer tree. *Each* tuner *is responsible for tuning the nodes in its attached subtree (denoted by shapes with similarly colored outlines). Right: One type of* tuner network *used in our experiments.*

6 Process Configuration

In this section we show each step of process configuration. At the same time we describe our implementation of the *reducer tree* and *tuner network* data structures.

6.1 Defining the Reducer Tree

A *reducer tree* can be constructed expediently using existing *geometry* and *material node* types. *Reducer trees* are not restricted to use only existing node types. New node types can be added by extending the `ReducerNode` class (see Figure 6). In particular, implementing a new *geometry node* type requires providing the function `getOutputIndex`. This function takes, as input, a 3D position and returns the `id` of its child (Geometry or Material) that contains this 3D point. This function also computes a local coordinate for this child node. Performing computations in local coordinates allows us to abstract away the geometry of a given object. Implementing a new `MaterialNode` type requires specifying the `getMaterial` function which takes a 3D point and returns the material at this point in the local geometry coordinate system. Both types of nodes have an `evaluate` function which is responsible for updating their internal states. This function is used by the *tuner network* to modify the internal state of the nodes. As an example, we show a *reducer tree* for performing texture mapping (see Figure 3). We also provide the corresponding pseudo-code (see Algorithm 1).

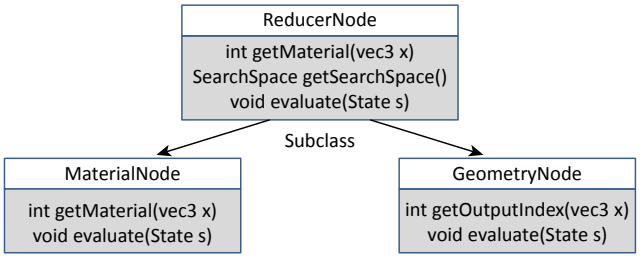

Figure 6: *Abstract interface for Node and its two subclasses.*

Algorithm 1 Constructing a *reducer tree* for texturing

```
1. Create a root node R from inputMesh
2. Subdivide R into outer layer O and inner
     volume V (Stratum Node)
3. Subdivide O into set of columns C
     (Column Node)
4. For each column c in C
5.   Subdivide c into two layers
       (Layer Node)
6. End
```

6.2 Defining a *Tuner*

Recall that a *tuner* consists of four components: a simulation, an error metric, an optimizer and a goal (Figure 4). Certain combinations of goal, metric and simulator are not compatible (i.e., a deformation simulator is not compatible with an error metric that compares images). Our API checks and prevents such incompatible combinations of components. The optimization algorithm can request the error value for a given state using a callback function `getError()` which is defined by the *tuner*. Additional callback functions can be defined by the developer depending on the needs of the algorithm. For example, the branch and bound algorithm requires a custom function to compute error bounds for a given state.

6.3 Binding *Tuners*

Tuners are assigned to nodes in the *reducer tree* using the `setNode` function. Once assigned, a *tuner* can optimize the parameters of its associated subtree. *Reducer nodes* provide a `getSearchSpace` function which returns all free variables in the node subtree. In order to make *tuners* as flexible as possible we provide a `Parameter` class. Parameters can be either discrete or continuous, they can have associated bounds, and they can be marked as free or fixed.

6.4 Establishing the *Tuner Network*

The *tuner network* is an undirected graph that describes connections between *tuners*. *Tuner nodes* store a list of their neighbors. Only neighboring *tuners* are allowed to exchange information. In our current implementation, this is accomplished using a shared memory array. As an example, we show how to construct and initialize a *tuner network* for a simple optimization scheme – a Simulated Annealing algorithm [van Laarhoven and Aarts 1987] (see Algorithm 2). The *tuner network* also requires a schedule that specifies in what order individual *tuners* should be executed. This schedule is specified by the developer. Once the *tuner network* is constructed, the process configuration phase is complete. We obtain a compiled executable that computes desired material assignment from an input specification.

Algorithm 2 Connecting and executing the *tuner network*

```
1.  for each Tuner T_i attached to a plane
2.    for each Tuner T_j adjacent to T_i
3.      add T_j to T_i's list of neighbors
4.    end
5.  end
6.  iterate N times
7.  for each Tuner T_i
8.    set temperature for T_i's optimizer
9.    run T_i
10. end
```

7 Process Use

The compiled program, which executes the *tuner network*, takes five types of arguments: the input geometry, the goal, the simulation configuration, a set of materials and the target 3D printer specification. After the *tuner network* is executed, the parameters of the *reducer nodes* in the *reducer tree* are set. It is then straightforward to compute the material assignment at arbitrary resolution. Since, a typical multi-material 3D printer requires a volumetric model with per voxel material assignment, we can simply iterate over all voxels in the volume. We obtain the material assignment by evaluating (`getMaterial`) of the *reducer tree* root at the center of each voxel location. This representation can be easily converted to a printer specific format for output. For the Objet500 Connex printer used in this paper we extract material isosurfaces which are submitted to the printer as STL files.

8 Results and Discussion

In order to evaluate capabilities of our system, we have implemented a number of existing translation processes. Furthermore, the ease with which different algorithm components can be combined enabled creation of two new translation processes. The first is

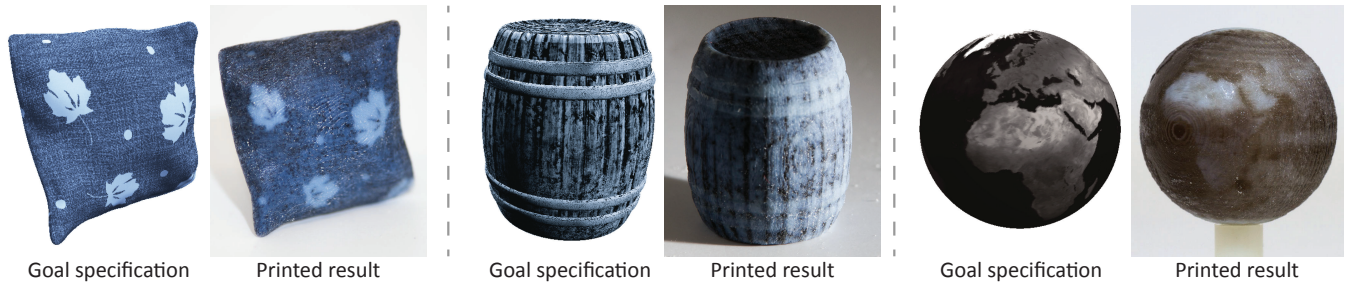

Figure 7: *The reducer-tuner model enables creating objects of arbitrary shapes with embedded textures. All results in this figure were created using the same reducer tree.*

an algorithm that can produce objects with desired refractive properties and an associated texture (Figure 1, right). The second allows us to apply a desired texture to an object with prescribed deformation behavior (Figure 1, left). All of these processes, both from prior work and new ones, easily fit into our framework, and were manufactured using a Stratasys Object500 Connex multi-material printer. The following paragraphs provide a detailed description of how the individual algorithms were designed with our system. All *reducer trees* and the *tuner networks* used for producing these results are presented in Figure 11.

Spatially-varying Albedo: We have designed a Spec2Fab translator that allows for 3D printing of textured models (Figure 7). Being able to apply precisely specified spatially-varying albedo values to printed models is a crucial capability. However, no standard processes have been designed for this task so far. The input to our algorithm is a shape and its desired albedo texture. Since the texture is only affected by materials close to the surface, we use a *stratum node* to divide the input shape into a thin shell and an inner volume. We then divide the outer layer into columns. The set of printable colors is expanded by stacking translucent materials using the `LayeredMaterial` Node. The number of layers is fixed to the number of print materials. The reduced parameters are the thickness values of each material layer. For each column, the *tuner*'s optimizer looks up the proper stacking that produces the closest albedo value. Due to printer resolution, the range of albedo values that can be achieved is quantized. We therefore implement an error diffusion algorithm by connecting neighboring *tuners*. In this simple algorithm, the simulation is a table lookup – we measure the albedo value corresponding to different base materials.

Heterogeneous Subsurface Scattering: We have replicated the subsurface scattering process of Hašan et al. [2010] using our framework (Figure 8). The input to our algorithm is a 3D mesh along with subsurface-scattering profiles defined at a set of surface points. We use the same *reducer tree* as in the texture example thereby simplifying the process configuration phase. The only difference is that we allow each column to have four layers of varying thickness and material. We use a branch and bound optimization algorithm which has been modified to handle continuous parameters by allowing discrete increments. We implement a bound estimate callback function specific to this problem. Each column is optimized independently using the algorithm. The simulation computes a scattering profile for a given stacking and the error metric compares the simulated and goal profiles using squared error.

Goal-based Caustics: We have configured two different processes for computing goal-based caustics (Figure 9). While they define exactly the same goal, they have very different *reducer trees* and *tuner networks*. The first process is based on the work of Papas et al. [2011]. It computes a set of micro-lenses which produce the desired caustic image. The image is pre-processed into a set of Gaussian distributions. Each distribution is matched with a microlens. The optimization applies simulated annealing to permute the location of these micro-lenses in order to construct a smooth surface. In the *tuner network*, each *tuner* is connected to its four neighbors. During the execution of an individual *tuner*, the optimization algorithm makes a randomized decision about whether or not to swap its micro-lens with one of its neighbors based on smoothness of the surface. The *tuners* are executed many times until a user-specified convergence criterion is met. In our examples, we use 45^2 microlens and ran all the *tuners* for 2000 iterations. In the second process, based on the work of Finckh et al. [2010], we use a *B-spline node* to represent a smooth caustic surface. This is in contrast to the potentially discontinuous surface in the method above. The reduced parameters are the height of each spline control point. We implement a simple caustics simulator for height fields. The simulated image is compared to the goal image using mean squared error. We used 60×60 control points and render the simulated image at 240×240 pixel resolution.

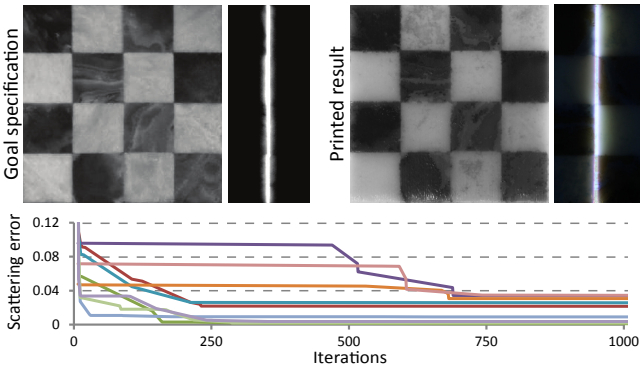

Figure 8: *A marble chessboard with prescribed subsurface scattering properties. The insets show the samples under thin line illumination, and the graph shows the convergence of tuners for 10 out of 100 scattering profiles used in the example. The error is measured by square distance between two profiles, each containing 400 coefficients.*

Elastic Behavior: In the spirit of Bickel et al. [2009] we have implemented an algorithm to compute material distribution based on a desired force-displacement response. The input to this algorithm is a mesh, a simulation configuration, and a desired shape. Simulation configuration includes vertex constraints and forces applied to the mesh. For this example, we use a co-rotational finite element method (FEM) simulation with linearly elastic materials to estimate the objects deformation. For the *reducer tree*, we use a voxel partition to divide the object into a low-resolution grid. We assign a

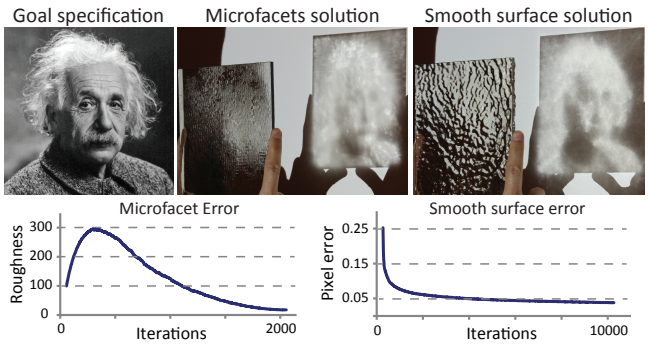

Figure 9: *3D printed lens arrays that each produce a caustic image of Einstein. Below we show convergence plots for both the microfacet and smooth surface optimizations. The portrait is available from the United States Library of Congress's Prints and Photographs division, now in the public domain.*

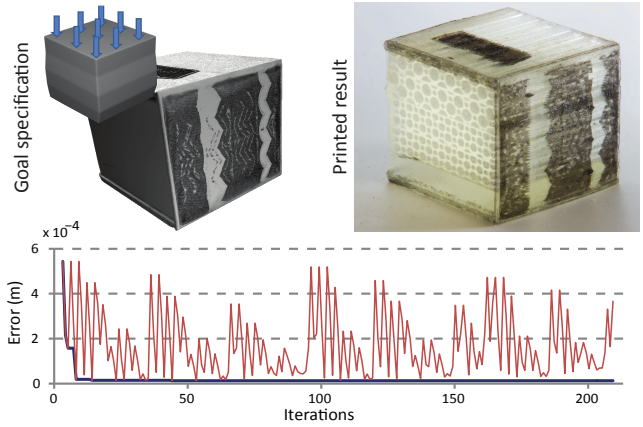

Figure 10: *A 3D printed book with prescribed deformation behavior under load. The plot shows the error (in meters) as a function of iteration number of our branch and bound based* tuner. *The blue line shows the smallest error seen so far while the red line the* tuner*'s progress exploring material subtrees [Bickel et al. 2009].*

single material to each grid cell. The FEM simulator queries the *reducer tree* for material assignments at arbitrary spatial locations. We use the same branch and bound algorithm as in our subsurface-scattering process but with a different bound computation callback function. We use the mean squared distance between the simulated and the desired shapes as the error metric. We have designed a simple experiment to validate our process in which we set the goal of our optimization to be a given deformed state (Figure 10).

Combining Caustics and Spatially-varying Albedo: The first of our new combined translation processes incorporates both smooth caustics and texture mapping (Figure 1, right). More specifically, we compute a transparent slab with a texture image that, when illuminated, casts a prescribed caustic image. The input slab is split into two pieces using a *plane node* as shown in Figure 11. The top piece is tuned to produce an input image. The material in the top piece is then fixed. The bottom piece is then tuned to produce a caustics image.

Combining Deformable Object and Spatially-varying Albedo: Our second new process combines spatially-varying albedo and elastic deformation properties (Figure 1 left). This is a very useful combination since when modeling objects we would like to specify both their appearance and "feel". In this process, the input shape is divided into a thin shell and an inner volume. We optimize for the deformation behavior and the texture independently due to the limitations of our current FEM simulator. As the outer shell is very thin, it has negligible influence on overall object deformation.

Discussion: Through our series of experiments, the *reducer tree* and *tuner network* have allowed us to reuse a large number of software components. In the extreme case, we arrive at our subsurface scattering algorithm by trivially adapting the same *reducer tree* for texture mapping objects. The power of component reuse is further elucidated by the presence of *column nodes* in most examples and by our ability to reuse optimization schemes (such as branch and bound) in multiple algorithms. The *reducer tree* and *tuner network* make these similarities easy to observe and exploit. The only component whose reusability was not demonstrated in this paper is simulation. In this work, we have aimed to fabricate objects with a wide range of physical properties and this necessitates the use of a wide range of simulation algorithms. Finally, once a whole process is configured, it is independent of input geometry and goal parameters. For example, we have run the spatially-varying albedo process with different geometries and input textures

The *reducer-tuner model* is ideally suited for multi-material printing capable of producing objects with a wide range of different properties. In order to showcase these strengths, we have fabricated our examples for an Objet500 Connex – a phase change inkjet printer that uses photopolymers with a wide range of optical and mechanical properties. Even though only two materials can be used and mixed within a single object, our framework has been proven to be very useful. As the number of materials that can be printed simultaneously will grow, we expect the methodology presented here to increase in utility. For other types of 3D printing technologies, Spec2Fab framework has a reduced use. In the case of 3D printing using a single, rigid material (e.g., fused filament fabrication or Stereolithography) the framework can be used to tune geometry of objects. For plaster-based 3D printers that produce full-color 3D prints, Spec2Fab framework can be employed to compute proper texturing of objects.

9 Conclusions and Future Work

In this paper we have taken the first step towards solving an open problem in computational fabrication – creating a general translation process that transforms user-defined model specifications into printer and material-specific representations. Our process relies on two data structures to make this general translation process expressive and computationally tractable: a *reducer tree* and a *tuner network*. We have shown how existing instances of this translation can be expressed and combined within our system. We believe that our API and its reference implementation will simplify and encourage development of new translation processes.

Our framework offers many exciting opportunities for future work. First, it would be extremely useful to implement many additional simulators in order to allow computing a variety of other properties, e.g., structural soundness, stability, material cost, and printing time. These simulators could be employed to expand the range of possible user-defined specifications. Similarly, only relatively simple error metrics have been tested within the tuning process. The development of more sophisticated and, in particular, perceptually-driven material metrics remains a relatively unexplored research area. Finally, it would be very beneficial to couple our API with a visual interface to further simplify the task of translator construction. It is not obvious what the best visual interface for specifying functional or physical properties of these objects is and more research in this area is necessary.

Acknowledgements The authors would like to thank Hanspeter Pfister, Sylvain Paris, Fredo Durand and Ilya Baran for their helpful suggestions as well as Bernd Bickel, Moritz Bächer and Miloš Hašan for providing software and data. This research was supported by the following sources of funding: DARPA #N66001-12-1-4242, NSF CCF-1138967, NSF IIS-1116296 and a Google Faculty Research Award.

References

ALEXA, M., AND MATUSIK, W. 2010. Reliefs as images. *ACM Trans. on Graphics (SIGGRAPH 2010) 29*, 4 (July), 60:1–60:7.

BARAN, I., KELLER, P., BRADLEY, D., COROS, S., JAROSZ, W., NOWROUZEZAHRAI, D., AND GROSS, M. 2012. Manufacturing layered attenuators for multiple prescribed shadow images. *Computer Graphics Forum 31*, 2 (May), 603–610.

BERMANO, A., BARAN, I., ALEXA, M., AND MATUSIK, W. 2012. ShadowPIX: Multiple images from self-shadowing. *Computer Graphics Forum 31*, 2 (May), 593–602.

BICKEL, B., BÄCHER, M., OTADUY, M. A., MATUSIK, W., PFISTER, H., AND GROSS, M. 2009. Capture and modeling of non-linear heterogeneous soft tissue. *ACM Trans. on Graphics (SIGGRAPH 2009) 28*, 3 (July), 89:1–89:9.

BICKEL, B., BÄCHER, M., OTADUY, M. A., LEE, H. R., PFISTER, H., GROSS, M., AND MATUSIK, W. 2010. Design and fabrication of materials with desired deformation behavior. *ACM Trans. on Graphics (SIGGRAPH 2010) 29*, 4 (July), 63:1–63:10.

BICKEL, B., KAUFMANN, P., SKOURAS, M., THOMASZEWSKI, B., BRADLEY, D., BEELER, T., JACKSON, P., MARSCHNER, S., MATUSIK, W., AND GROSS, M. 2012. Physical face cloning. *ACM Trans. on Graphics (SIGGRAPH 2012) 31*, 4 (July), 118:1–118:10.

COOK, R. L. 1984. Shade trees. *Computer Graphics (SIGGRAPH 84) 18*, 3 (Jan), 223–231.

CUTLER, B., DORSEY, J., MCMILLAN, L., MÜLLER, M., AND JAGNOW, R. 2002. A procedural approach to authoring solid models. *ACM Trans. on Graphics (SIGGRAPH 2002) 21*, 3 (July), 302–311.

DONG, Y., WANG, J., PELLACINI, F., TONG, X., AND GUO, B. 2010. Fabricating spatially-varying subsurface scattering. *ACM Trans. on Graphics (SIGGRAPH 2010) 29*, 4 (July), 62:1–62:10.

FINCKH, M., DAMMERTZ, H., AND LENSCH, H. P. A. 2010. Geometry construction from caustic images. *Computer Vision (ECCV 2010) 6315*, 464–477.

HAŠAN, M., FUCHS, M., MATUSIK, W., PFISTER, H., AND RUSINKIEWICZ, S. 2010. Physical reproduction of materials with specified subsurface scattering. *ACM Trans. on Graphics (SIGGRAPH 2010) 29*, 4 (July), 61:1–61:10.

JACKSON, T. R. 2000. *Analysis of Functionally Graded Material Object Representation Methods*. PhD thesis, Massachusetts Institute of Technology.

JORDAN, M., GHAHRAMANI, Z., JAAKKOLA, T., AND SAUL, L. 1999. An introduction to variational methods for graphical models. *Machine Learning 37*, 183–233.

KENNEDY, J., AND EBERHART, R. 1995. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, IEEE, 1942–1948.

KOU, X., AND TAN, S. 2007. Heterogeneous object modeling: A review. *Computer-Aided Design 39*, 4, 284 – 301.

KOU, X., PARKS, G., AND TAN, S. 2012. Optimal design of functionally graded materials using a procedural model and particle swarm optimization. *Computer-Aided Design 44*, 4, 300–310.

KUMAR, V., BURNS, D., DUTTA, D., AND HOFFMANN, C. 1999. A framework for object modeling. *Computer-Aided Design 31*, 9, 541 – 556.

LOW, Y., GONZALEZ, J., KYROLA, A., BICKSON, D., GUESTRIN, C., AND HELLERSTEIN, J. M. 2010. Graphlab: A new parallel framework for machine learning. *Conference on Uncertainty in Artificial Intelligence (UAI)* (July).

MALZBENDER, T., SAMADANI, R., SCHER, S., CRUME, A., DUNN, D., AND DAVIS, J. 2012. Printing reflectance functions. *ACM Trans. on Graphics (SIGGRAPH 2012) 31*, 3 (May), 20:1–20:11.

MATUSIK, W., AJDIN, B., GU, J., LAWRENCE, J., LENSCH, H. P. A., PELLACINI, F., AND RUSINKIEWICZ, S. 2009. Printing spatially-varying reflectance. *ACM Trans. on Graphics (SIGGRAPH Asia 2009)) 28*, 5 (Dec.), 128:1–128:9.

MITRA, N. J., AND PAULY, M. 2009. Shadow art. *ACM Trans. on Graphics (SIGGRAPH Asia 2009) 28*, 5, 156:1–156:7.

PAPAS, M., JAROSZ, W., JAKOB, W., RUSINKIEWICZ, S., MATUSIK, W., AND WEYRICH, T. 2011. Goal-based caustics. *Computer Graphics Forum 30*, 2, 503–511.

PAPAS, M., HOUIT, T., NOWROUZEZAHRAI, D., GROSS, M., AND JAROSZ, W. 2012. The magic lens: refractive steganography. *ACM Trans. on Graphics (SIGGRAPH Asia 2012) 31*, 6 (Nov.), 186:1–186:10.

SKOURAS, M., THOMASZEWSKI, B., BICKEL, B., AND GROSS, M. 2012. Computational design of rubber balloons. *Computer Graphics Forum 31*, 2 (May), 835–844.

VAN LAARHOVEN, P., AND AARTS, E. 1987. *Simulated annealing: theory and applications*, vol. 37. Springer.

VIDIMČE, K., WANG, S.-P., RAGAN-KELLEY, J., AND MATUSIK, W. 2013. OpenFab: A programmable pipeline for multi-material fabrication. *ACM Trans. on Graphics (SIGGRAPH 2013) 32*, 4 (July).

WEYRICH, T., PEERS, P., MATUSIK, W., AND RUSINKIEWICZ, S. 2009. Fabricating microgeometry for custom surface reflectance. *ACM Trans. on Graphics (SIGGRAPH 2009) 28*, 3 (July), 32:1–32:6.

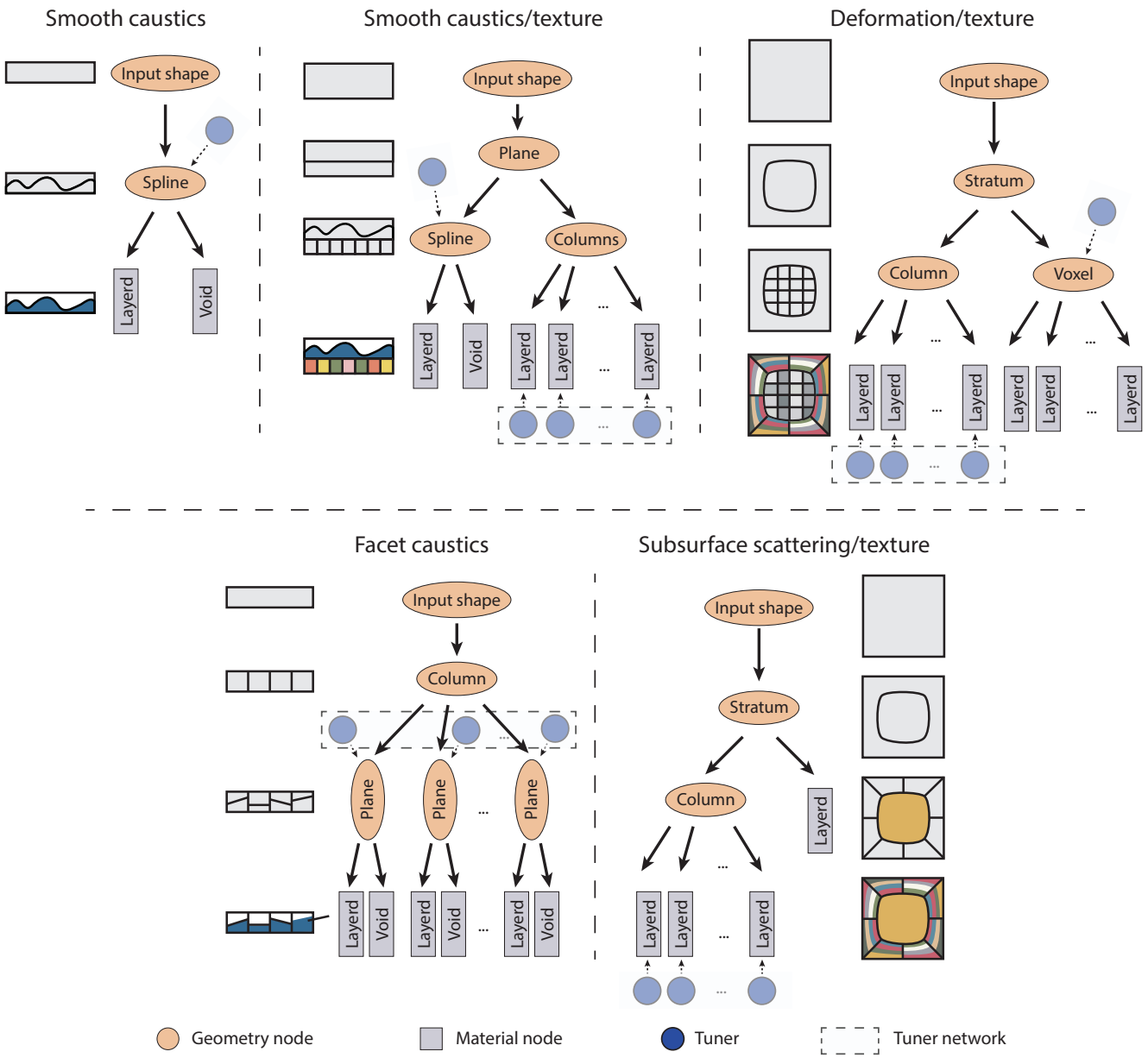

Figure 11: *The figure shows all* reducer trees *used in this paper.* ***Smooth caustics:*** *The smooth caustics tree consists of a spline surface that divides the geometry into an inner surface filled with a transparent material and an outer, void surface. This allows us to approximate displacement mapping. Here, a single* tuner *is used deform the spline surface.* ***Smooth caustics/texture:*** *The smooth caustics/texture* reducer tree *splits the object into upper and lower sections using a plane. The upper section then uses the Smooth caustics* reducer tree *to define surface displacements while the lower section uses a set of columns as texture pixels. Here, a single* tuner *solves for the caustic image and a set of* tuners *with regular connectivity solves for the textured image.* ***Deformation/texture:*** *The object is divided into an outer layer and inner volume using a stratum node. The interior volume is then voxelized and each voxel is assigned a layered material. As in the previous example, the outer layer is divided into columns which are used as pixels for the texture image. The* tuner *setup is also similar to the previous example except the single* tuner *is used to optimize the mechanical behavior of the object.* ***Facet caustics:*** *Here, we build facets by dividing an object into columns and then bisecting the columns horizontally with a plane. The orientation of this plane defines the angle of the facet surface. We use a separate* tuner *to optimize each plane orientation.* ***Subsurface scattering/texture:*** *The subsurface scattering/texture* reducer tree *partitions the object into an inner volume and outer layer using a* stratum node. *The inner volume is unimportant in this example so we place a constant material there. The outer material is divided into pixels using the* column node. *These columns are filled with layered materials. We optimize each column separately using an associated* tuner.