Style Grammars for Interactive Visualization of Architecture

Daniel G. Aliaga, Paul A. Rosen, and Daniel R. Bekins
Department of Computer Science at Purdue University

Abstract—Interactive visualization of architecture provides a way to quickly visualize existing or novel buildings and structures. Such applications require both fast rendering and an effortless input regimen for creating and changing architecture using high-level editing operations that automatically fill in the necessary details. Procedural modeling and synthesis is a powerful paradigm that yields high data-amplification and can be coupled with fast rendering techniques to quickly generate plausible details of a scene without much or any user interaction. Previously, forward generating procedural methods have been proposed where a procedure is explicitly created to generate a particular content. In this article, we present our work in inverse procedural modeling of buildings and describe how to use an extracted repertoire of building grammars to facilitate the visualization and quick modification of architectural structures and buildings. We demonstrate an interactive application where the user draws simple building blocks and using our system can automatically complete the building "in the style of" other buildings using view-dependent texture mapping or non-photorealistic rendering techniques. Our system supports an arbitrary number of building grammars created from user subdivided building models and captured photographs. Using only edit, copy and paste metaphors, entire building styles can be altered and transferred from one building to another in a few operations, enhancing the ability to modify an existing architectural structure or to visualize a novel building in the style of others.

Index Terms— Display Algorithms, Image-based Rendering, Modeling Packages, and Visualization systems and software.

1 INTRODUCTION

The interactive visualization of architecture and buildings provides a way to see current structures as well as future tentative structures and changes to existing buildings. The input regimen, display algorithms, and technology should ideally be intuitive, fast and realistic, and transparent to the user. Computing power and graphics technology has advanced significantly in recent years supporting the fast rendering of complex architectural scenes.

However, a common design challenge for such interactive visualization applications is to require little effort by the user to create or to alter architecture and buildings. Ideally, the system must be able to create interesting and consistent alterations to the structures from only a few specifications on part of the user. Thus, an interactive visualization program must include a component that at least semi-automatically infers details of the structures being observed and changed. The inferred details are then used to fill-in the structures and produce interactive renderings of potentially new architectural structures.

Procedural modeling and synthesis is a powerful paradigm that can be coupled with an interactive rendering program for architecture to generate plausible details of a model without much or any user interaction. Procedural methods have the advantage of exhibiting a high-degree of detail amplification; e.g. using only a small number of parameters, significant plausible details can be synthetically generated. However, since a small change in the parameters can cause huge changes in the resulting model, it is extremely difficult to determine a good set of procedures and parameters. Nonetheless, promising results have been demonstrated in several restricted arenas such as fractal-based compression and L-systems for procedurally generating plants [1]. Furthermore, pre-specified grammars have been used to generate plausible cities [2] and some architectural structures [3, 4].

In this article, we describe an interactive system that enables both creating new buildings in the style of others and modifying existing buildings in a quick and intuitive manner. In a first step, our system provides tools to the user for mapping photographs of an existing building to a simple geometric model and for subdividing a building into its basic external features (e.g., floors, windows, doors, trim, brick, wood, etc). In a second step, the system automatically creates a representative grammar that captures the repetitive patterns and particularities present in the building and its features. This grammar essentially captures the style of the building and enables us to transfer the style to new un-subdivided models of potentially very different shapes. In a third and interactive step, users draw a desired new building configuration using simple building blocks and the system uses the grammar to automatically subdivide the new building configuration. This results in a new and complete building or a modified version of a captured building, both in the style of the original.

The new building can be rendered using a view-dependent texture mapping of image fragments from the captured photographs, or using a stylized procedural rendering (e.g., pen-and-ink) of the terminals of the grammar. Moreover, the redundancy among the images of the building can also be used to automatically fill occluded and poorly sampled areas of the image set, as well as to equalize the color and lighting between images and surfaces of the model.

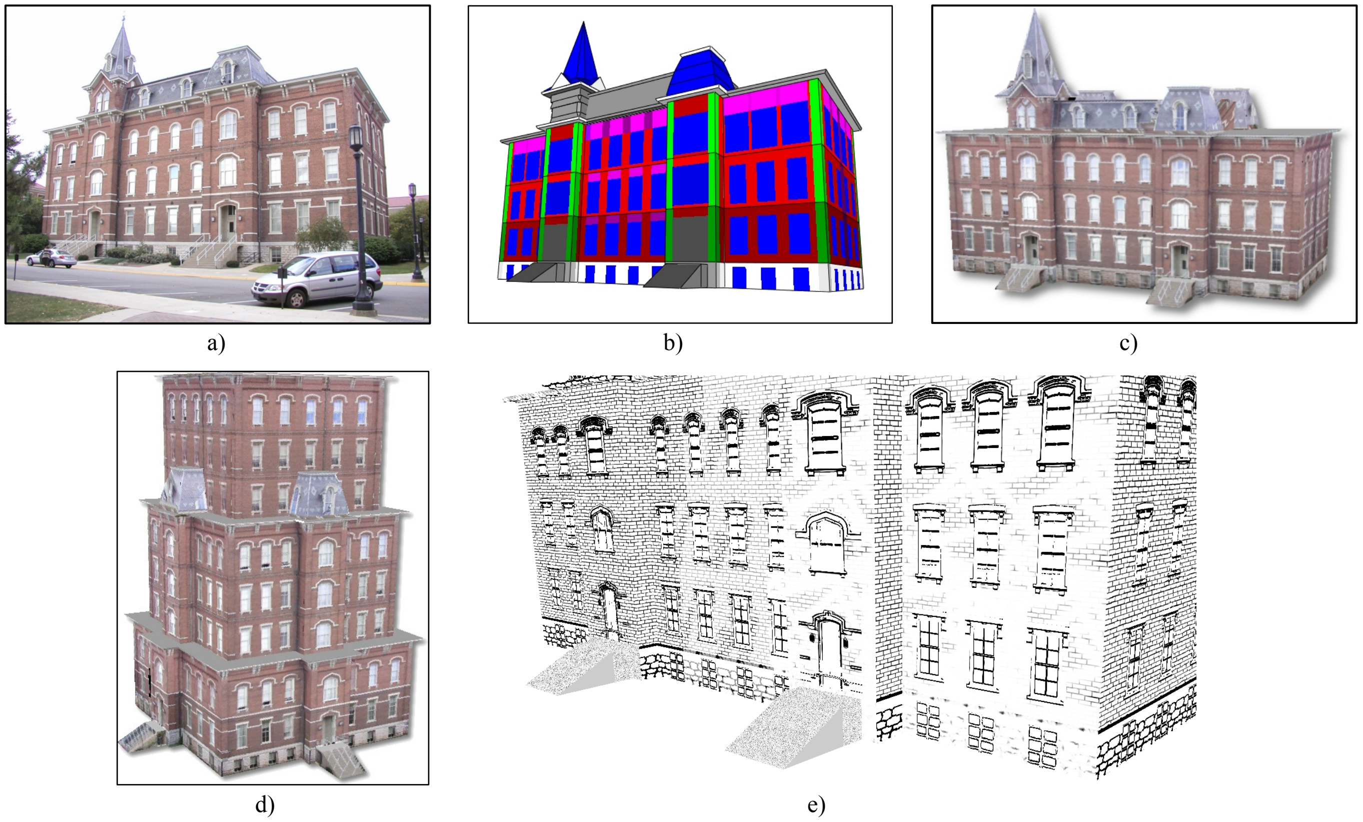


Figure 1. System Overview. *Our system enables users to modify and render architectural structures based on grammars extracted from various real-world buildings. First, the user creates and subdivides an initial model of the building (a-b), then our algorithm automatically finds repetitive patterns of the building features and constructs a representative grammar. Using an interactive program, the user can then view the captured model (c), change the model on the fly producing new models (d) and view stylized renderings (e).*

Finally, we include tools to specify trees, ground, and sky yielding a complete architectural scene in minutes yet supporting completely changing the style of the buildings at any moment. The user can instantaneously copy and paste new building-styles and apply most affine transformations, including uniform and non-uniform scaling, to the building configurations. The grammar is interactively reapplied and continuously adjusts to the size and shape of the building configuration. We present the results of using our system to sketch, modify, and render new buildings created in the style of existing real-world buildings.

The contributions of our work include

- an algorithm to construct a grammar from a subdivided building model and to use the grammar to fill-out new building configurations with the captured architectural details,
- a system to interactively create new buildings in the style of existing real-world buildings using view-dependent texture mapping,
- a method to create and interactively edit non-photorealistic stylized renderings of novel and existing architectural structures, and
- rendering techniques for drawing new buildings omitting occlusions in the original data and equalizing lighting and shading.

2 Previous Work

The modeling and rendering of 3D objects and architectural structures has been addressed in several ways in computer graphics. Photogrammetric reconstruction and image-based modeling and rendering (IBMR) build models from real-world photographs. Procedural modeling focuses on generating synthetic models of objects and environments from a pre-specified set of rules and terminals defined by a grammar. A few interactive sketching systems have been proposed for the inverse procedural modeling of plants. Our work builds upon these areas of research to develop a novel method for architectural visualization.

Photogrammetric reconstruction and IBMR build a representation of the observed objects and enable mapping the acquired image data to the representation. In particular, Facade [5] has served as the prototype for several commercial packages that reconstruct an approximate geometric model with user-assistance and then texture map view-dependent images [6] onto the model (e.g., [7, 8, 9]). Image-based modeling and rendering is a partner of capture techniques like photogrammetric modeling, but unlike photogrammetric modeling, an IBMR system directly re-samples photographs of a static scene to create novel views of the acquired object [10, 11, 12, 13] or small environment [14, 15]. Some efforts have focused on reconstructing large urban spaces [16] or on reconstructing buildings in particular

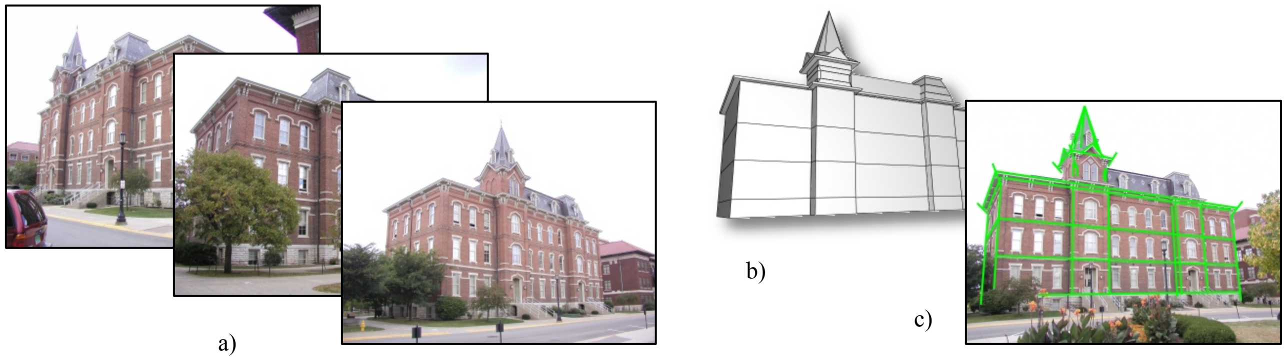


***Figure 2. Building Specification**. (a) Photographs of the building are taken. (b) A simple model of the scene is created. (c) Model edges are matched to image edges. The model parameters and image parameters are computed automatically via an optimization process.*

from images [17]; however, not on quickly creating novel buildings, editing existing buildings, and transferring style.

Procedural modeling is useful for creating objects that exhibit a high-degree of redundancy. For example, L-systems have been successful in the modeling of plants [1] and have been used for automatic city and building generation [2, 3]. Shape grammars, which define rules for the specification and transformation of 2D and 3D shapes [18], have also been used to model architecture. Wonka et al. [3] and Mueller et al. [4] let a user specify parameters and employ a pre-specified grammar to automatically generate buildings and architecture from a database of given rules and attributes. Legakis et al. [19] use cellular patterns to create façade-level details for architectural models. While procedural modeling provides a means for quickly creating architecture from a small number of terminals and rules, the data and procedures are not extracted from an actual real-world city or building.

Some drawing systems have been proposed for the inverse modeling of plants and trees. As opposed to forward-generating procedural plant sketching systems, these systems infer structure from observed data and build a model. For example, Shlyakhter et al. [20] build models of trees by fitting a coarse branching proxy to a set of instrumented photographs. On the other hand, in our work, we wish to infer a grammar for a non-organic structure, such as a building, based on acquired photographs.

Several general and focused sketching systems have been presented in the literature for quickly drawing new objects. For instance, Zeleznik et al. [21] and Shesh and Chen [22] combine synthetic rendering with a pen-based interface to create geometry, including simple architecture. Oh et al. [23] and Google Sketchup [24] have developed tools for sketching buildings. However, these structures are created manually and not from existing architecture.

We seek an interactive visualization application that starts with simple building blocks and automatically fills out the details of a building with a chosen style. The styles and corresponding grammars are obtained from images of real-world buildings. While some previous work has created grammars for individual building facades [25], our focus is on quickly generating modifications to entire buildings or creating new buildings similar to others.

The work presented in this article is an extension of our conference publication [26]. However, in this article, we generalize the system to style grammars composed of a hierarchy of production rules. This provides significant additional flexibility, including the ability to copy-and-paste arbitary subsets of the grammar and to rearrange the rules within the hierarchy. We also present semi-automatic methods to convert terminals to stylized renderings similar to pen-and-ink, provide interactive tools for creating landscapes (in [26], landscapes were created non-interactively) and give a complete description of a system for interactively visualizing existing and new buildings. Our system greatly simplifies the visualization of architecture, enables fast content creation and editing, and supports instantaneously applying entirely new building styles.

3 Building Grammars

A typical building contains a regular structure that can be exploited to automatically detect patterns in its configuration and construct a representative grammar. A building consists of several floors; each floor is divided into various faces and each face consists of several windows surrounded by trim and wall material. Within a single building there can be groups of differently shaped floors, a variety of window styles and trims, and several types of wall material such as brick, stone, etc. Our algorithm exploits this typical global structure and captures the significant local details. In this section, we first describe how to incorporate a new building into the system. Then, we create the grammar and derive new instantiations enabling the modification of existing buildings or the creation of new ones.

3.1 User Specification

To add a new real-world building to the system, the user must map edges in captured images to a scene graph subdivided into basic building features. The scene graph creation and edge mapping is done once using a graphical user interface (GUI). No attention need be paid to the dimensions of the scene or camera pose, as these will be recovered by the system via an optimization. This specification provides information that will be used to automatically detect patterns in the structure of the captured building.

Our system provides tools to quickly build a model from a collection of photographs (Figure 2a) and from parameter-

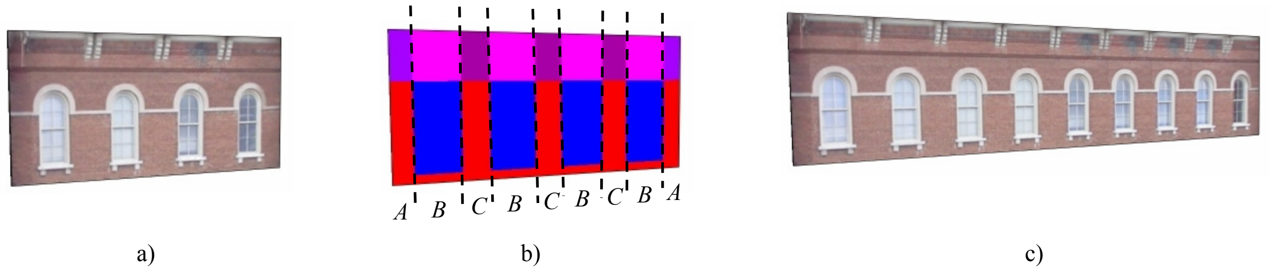


***Figure 3. Face Productions**. Patterns detected in a face are used to infer a grammar and build new faces. (a) An original face from a captured building. (b) A subdivision of the face yielding grammar F = ABCBCBCBA = A(BC)*BA. (c) A new stretched face filled-out automatically using the inferred production rule.*

ized geometric building blocks organized into a hierarchical scene graph (Figure 2b). Each node of the graph contains a block, and each edge of the graph represents a transform specifying the position and orientation of a block relative to its parent or child. Each block is composed of a small set of vertices and a simple geometrical structure (e.g., box, cylinder, pyramid, etc.). The completed model will have the dimensions and pose of each block. Our system also supports constraining block size, position, or alignment in relation to other blocks. This reduces the total number of parameters and leads to a more robust optimization.

Once the model is created, it is matched to the image set. For each image, the user employs our GUI to mark the visible part of prominent image edges and map them to the corresponding model edges (Figure 2c). We do edge correspondence, as opposed to point correspondence, because they are more likely to be at least partially visible in a given image. In fact, the occluded regions of the buildings are marked as such and will be ignored. Also, it is by no means necessary to mark every visible edge. Starting with an undetailed model and only a few images and edges, the system incrementally improves camera pose estimation and structure recovery. To compute the scene parameters, we define an error function based on the discrepancy between the user-marked (observed) edges and the model edges as viewed by each camera. By minimizing this error function, we recover the parameter values that align the model edges most closely to the observed edges.

The user then subdivides the model into features and feature groups. Figure 1b shows a subdivision and labeling scheme for an example building where each feature group is rendered in a unique color. Block subdivision is performed to divide the building into floors. Surface subdivision is used to divide the building facade into labeled feature groups representing, for instance, brick, trim, windows, and entries. In addition, it is necessary to indicate whether each feature group is of a fixed size. For example, windows, door, and trim are of fixed sized, while brick regions are not fixed in size. This tells the system that the brick can be repetitively tiled or cropped on a novel face, while the windows and doors should remain the same size.

3.2 Grammar Parsing and Derivation

Using the subdivided model, the system automatically finds the repetition of building features and creates a collection of production rules and terminals that represent the captured building. For parsing, all buildings are assumed to be organized in the following manner. First, the blocks of a building specification are sorted from bottom to top. Unless otherwise specified, the first two subdivided blocks and the last subdivided block are considered the base, ground-floor and roof, respectively. The intermediate floors are labeled as repeatable floors. Second, each floor consists of an ordered sequence of faces and each face is formed by an array of columns. Third, a column may correspond to a single terminal or to a group of vertically stacked terminals.

This organization defines a hierarchy of production rules where the grammar terminals consist of images of basic building features (e.g., windows, trim, brick, stone, etc.). Thus, the grammar has the following general form:

$$\begin{array}{lll}
\texttt{Model } M & \rightarrow & \texttt{(base)(ground)}\{\ S_0\ S_1\ \dots\ S_{N-1}\ \}\texttt{(roof)} \\
\texttt{Floor } S & \rightarrow & \{\ F_1\ F_2\ \dots\ F_M\ \} \\
\texttt{Face } F & \rightarrow & \{\ C_1\ C_2\ \dots\ C_P\ \} \\
\texttt{Column } C & \rightarrow & \{\ T_1\ T_2\ \dots\ T_R\ \}.
\end{array}$$

Grammar parsing automatically obtains the particular instantiations of these general rules and terminals for a captured building. These rules describe the basic ways in which an element (e.g., model, floor, face or column) can grow, shrink, and be adapted to any given collection of building blocks. The process of deriving a completely new building entails determining which production rules to apply and how many times to repeat them. Thus, given the original un-subdivided building blocks or a new set of building blocks, the blocks can be automatically subdivided into floors, faces, columns, and terminals yielding a new and similar-in-style building. In the following sections, we describe how to create the instantations of these rules.

Face Productions

A face *F* of a captured building is represented by a production rule containing symbols for each individual column and geometric information for determining precisely how many repetitions of each column to use when creating a novel face of arbitrary size. Consider the face in Figure 3a which is subdivided into nine columns. Two columns in a face are considered similar if the labels of their respective terminals match one-to-one. Thus, the pictured face has only three unique types of columns and can be written as *F* = *ABCBCBCBA* (Figure 3b). Repetitions of similar groups of columns are combined and represented by borrowing the Kleene star notation from regular expressions. Hence, we write as a possible production rule "*F* → *A*(*BC*)**BA*", which

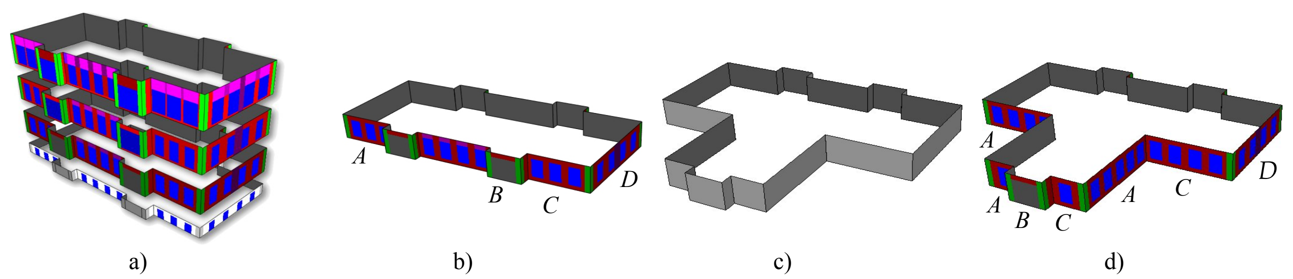


Figure 4. Floor Productions*. A model is divided into several floor surfaces that wrap around the building and can be applied to a floor of a novel model. (a) A captured building and its floors. (b) A captured model floor. (c) A novel model floor. (d) Face production rules from the captured floor applied to the novel floor.*

implies the pattern BC can be repeated an arbitrary number of times.

While many different forms of repetitive patterns can occur, our system searches for repeating pairs of elements. In practice, we have found this to yield fairly compact and flexible production rules. Having large repeating groups of elements makes it difficult to accommodate to small building configurations changes. Further, in order to keep approximately the same distribution of features, we ignore repetitions of the form AA. For example, if A represents brick and B corresponds to a window, then an approximately similar distribution of windows on a face occurs by repeating AB. On the other hand, repeating AA creates more brick and decreases the density of windows on a face of a building. Based on these observations, our algorithm obtains a compact production rule for face *F* by first scanning the string and marking each reoccurring pair of the form *AB* (but not *AA*). Then, adjacent repeated instances of a marked pair are replaced with a single instance and the Kleene star is added to all marked pairs.

Consider the following typical derivations produced by our system:

```
1.  ABABABA ⟶ (AB)(AB)(AB)A ⟶ (AB)*A
2.  AABAABAA ⟶ A(AB)A(AB)AA ⟶ A(AB)*A(AB)*AA
3.  ACABABA ⟶ AC(AB)(AB)A ⟶ AC(AB)*A
```

The first example represents a very common and straightforward situation where repeating pairs have been combined yielding a compact rule. If we derived the rule (*ABA*)**B*(*ABA*)* for this example, it would goes against the alternating *AB* pattern and would also make the rule less flexible by requiring elements of greater width to be squeezed into a face of arbitrary size. In the second example, the repeated pattern *AA* is not merged based on our observation. Otherwise this would have resulted in the rule (*AA*)**B*(*AA*)**B*(*AA*)*, which is likely to be much less visually interesting when stretched.

While it is certainly possible to detect higher-level patterns in a face and perform different groupings, obtaining rules that maintain coherent structure over multiple floors is difficult and thus should be imposed by the user through multiple levels of face subdivision. Consider the facades of two floors where pattern 3 is directly below pattern 1 (*C* might represent a door, while *B* represents a window). Using a higher-level rule for only pattern 1 (or pattern 3) might ruin the vertical coherence between the two floors by adding extra instances of columns to the associated floor of a novel building. Our set of rules does not attempt to infer high-level and multi-floor structure but, in practice, yields good coherence and compactness in a variety of situations.

To apply the production rule of face *F* to a novel and arbitrarily larger face *F′*, we must determine the number of repetitions of each repeatable column and a scale factor for columns of variable width. We calculate a common multiplier *k* for all repeating columns such that the remaining width is filled as much as possible without overflowing. Using a single common multiplier preserves the symmetry and balance of the face structure. The remaining width of *F*' is filled by adding at most one more repetition of each repeating column. The configuration that yields a scale factor for the variable width columns closest to one is chosen.

Mapping a face F to a novel and smaller face F′ yields two possible scenarios. If the new face F′ is larger than or equal to face F without any repeating patterns, then the same method from the previous paragraph can be used. Otherwise, we calculate a scale factor for the columns of variable width that makes the size of F equal to F′. If no such scale factor exists, we choose to omit the smallest features of F until it is smaller than F′ and then calculate the scale factor for the columns of variable width.

Figure 3c shows the results of the application of a face production rule to a novel face. The novel face remains true to the original style. Section 4 describes in more detail how the rendering of novel faces is accomplished.

Floor Productions

A floor production rule is a description of the outward facing surface of a single floor wrapping around a model. Each rule consists of several face productions $S = \{F_1, F_2, ..., F_M\}$, where F_i is connected horizontally to $F_{(i+1)\ mod\ M}$. Each face has exactly one left adjacency and one right adjacency. The corner orientation between adjacent faces is also recorded. It indicates whether the faces meet at an inner corner, outer corner, or are continuous. For example, the model in Figure 4a contains four floors, including a small base floor and three intermediate floors. Each floor surface wraps around several blocks and therefore contains several different and adjacent face productions and corner orientations.

Our method uses a set of criteria to apply a floor production rule to a floor of a novel building. A novel model contains the faces $S' = \{F'_1, F'_2, ..., F'_{M'}\}$. The system

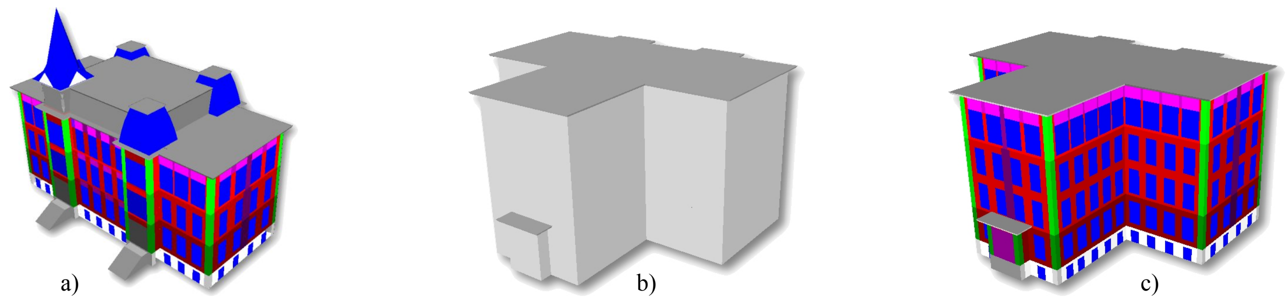


Figure 5. Model Productions. *The model productions from a captured model are applied to a novel model. (a) The captured model. (b) The novel model. (c) The novel model automatically subdivided in the style of the captured model.*

selects a production rule F_i for each novel face F'_j and then applies the selected rule to the face as described in the previous section. Our system uses a weighted combination of the following three criteria to determine the fitness of a candidate face production rule to a given novel face:

- Corner Orientation. A face is frequently characterized by the types of corners that surround it; for example, an outer corner is likely to have trim or decoration, while an inner corner is less likely to. Since each face has two adjacencies and there are three types of corners, this results in nine categories of faces. The best candidate production will match the novel face in this respect.
- Size. The candidate production that is closer in size to the novel face is more likely to be appropriate.
- Resolution. If two or more candidate productions are appropriate in terms of corner orientation and size, the production from the face with the highest quality image samples is selected.

Figures 4(b-d) show the results of applying a captured floor production rule to a novel floor. The capitalized letter labels indicate the choice of capture face for each novel face. Our approach does not consider the image-content of each terminal and thus mirrored-copies of terminals are not used. Nevertheless, it can be seen that using the corner orientation, size, and resolution of each face results in the terminals, including trim, being consistently applied.

Model Productions

A model production rule consists of all the aforementioned production rules and enables subdividing a given collection of building blocks in a single operation. The floors of the captured model are stored in a directed graph that usually resembles an array. However, in special cases such as a breezeway or towers, a single floor can be connected to multiple floors. Applying a model production rule to a new set of building blocks requires the system to determine multipliers for each repeating floor element and then align and subdivide each block accordingly. The blocks are first sorted in order of height. The tallest block will be used as the basis block for determining the number of repetitions for each floor. This is computed by determining the multipliers for each repeating floor that result in the closest match to the basis block height. The basis block and all its connected blocks are then resized and subdivided according to the model production rule that matches their vertical positions most closely. After subdivision, the floor surface connectivity is updated, and the appropriate floor production rules are applied to each floor of the novel model. All blocks that are attached to the basis block are marked, and the algorithm continues with the remaining blocks, if any.

Figures 5 and 1d shows the application of a captured model onto a novel model. It can be seen that the floors have been repeated multiple times in order to fill the entire height of the new building. The application of the model production rules occurs in a fraction of a second, for example, during a single copy-and-paste operation performed by the user.

4 Interactive Rendering

Our system uses view-dependent rendering strategies to portray the building structures in real-time. The application of the production rules generates the geometry of the building and its facades. In this section, we focus on drawing the terminals of the building grammar. Our approach enables photo-realistic rendering using color-equalized view-dependent projective texture mapping and non-photorealistic rendering using stylized terminals. Both of the rendering strategies are also provided with mechanisms to replace occluded regions of the captured structure.

Projective Texture Mapping

Our system can render a terminal using color-equalized view-dependent textures. A given terminal is seen in multiple images captured from different distances and angles. Shading and lighting effects will slightly change the captured colors of a terminal from image to image. Hence, the rendering method must approximately equalize the colors and brightness of the terminals and select the most appropriate samples for the terminals as seen from the current viewing distance and angle.

Color equalization is possible by comparing the color data from terminals of the same feature group in different locations on the model. During subdivision, the user indicates which terminal types are to be considered diffuse (e.g., brick, stone) and marks one or more of the captured images as color keys. Then, the program determines the average color of each diffuse group from the color key images. For each image, the average color of each diffuse group and for

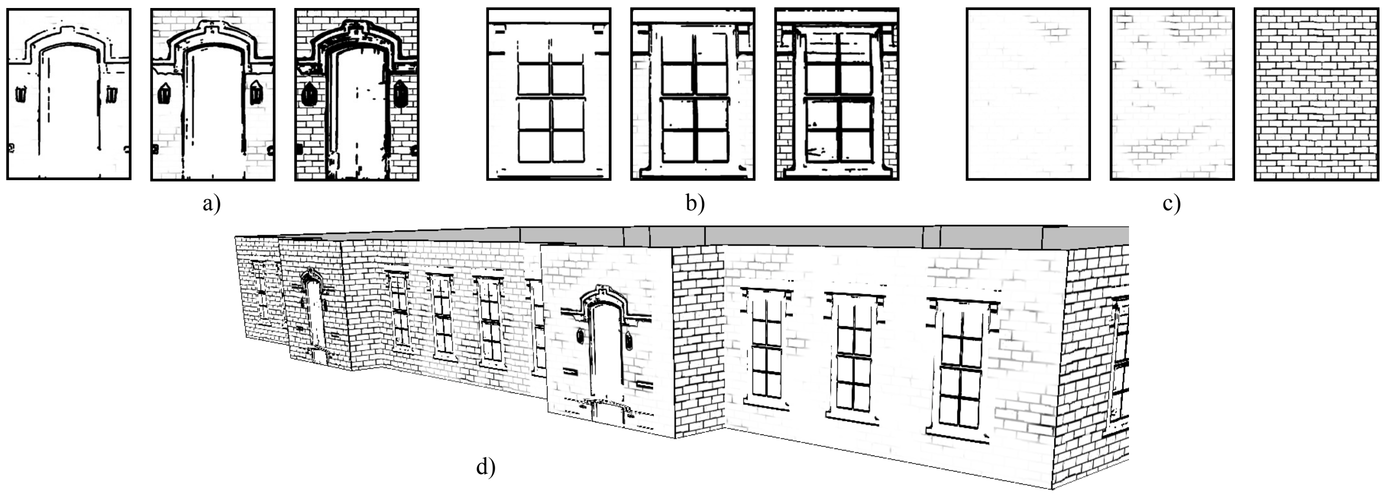


Figure 6. ***Stylization.*** *Our method supports procedural stylizations of the terminals of the inferred grammar. (a-c) Low-to-high tonal views of stylized terminals. (d) Example rendering of a novel floor of a captured building. Light source is in front of façade and near the observer; brighter stylizations are used to resemble highlights.*

each surface normal is computed. By averaging the colors for different surface normals, we equalize shading between surfaces as well as colors between images. Equalization is performed by color channel shifting. The shift amount for each surface in each image is computed as the difference between the surface average color and key average color.

To render each terminal, the system blends between the most similar sampled views. In particular, the viewpoint of each image is transformed into a coordinate space local to the terminal and projected onto the unit sphere in this space. During each frame of rendering, a lookup function determines the three source views closest to the current view in distance and in angle [3]. These views are weighted according to distance and angle to give smooth transitions during interactive rendering (Figures 1d and 7d).

Stylization

The partitioning of the building into discrete terminals also enables several forms of stylized rendering (e.g., [27, 28, 29]). Our system produces interactively editable illustrations of buildings drawn in pen-and-ink style [30]. Prior to rendering, our system uses a semi-automatic stylization procedure to pre-render multiple versions of each terminal. The cached images span both resolution-space and tonal space affording near and far views of the buildings and providing stylized shading by smoothly varying the apparent brightness of each terminal.

During grammar creation, the system identifies from among the multiple views of each terminal a best image sample for each and creates a cache of stylized images of all terminals. To the best image samples, we apply a filter yielding a thresholded binary image with edges detected. Foreground pixels are connected to form a geometrical mesh rendered in black over a white background. To create images at multiple resolutions but at the same tonal level, the mesh is drawn using a line width chosen so that the average per-unit area intensity matches that of the original image. Terminal images are also generated at different tonal levels by using random strokes to "wash out" parts of the image and produce light/dark regions (Figure 6a-c). Since some terminals are marked as being of variable size during model subdivision, the corresponding stylized terminals are made repeatable as well via mirroring and blending. At runtime, we select for each terminal the best tonal level using a diffuse shading model with distance-based attenuation.

Figure 6d shows example stylizations for a novel stretched floor. Since our terminal rendering is procedural, we can generate crisp and varying tone images for all terminals, include repeated terminals, and produce a smooth and visually-pleasing stylization at any resolution.

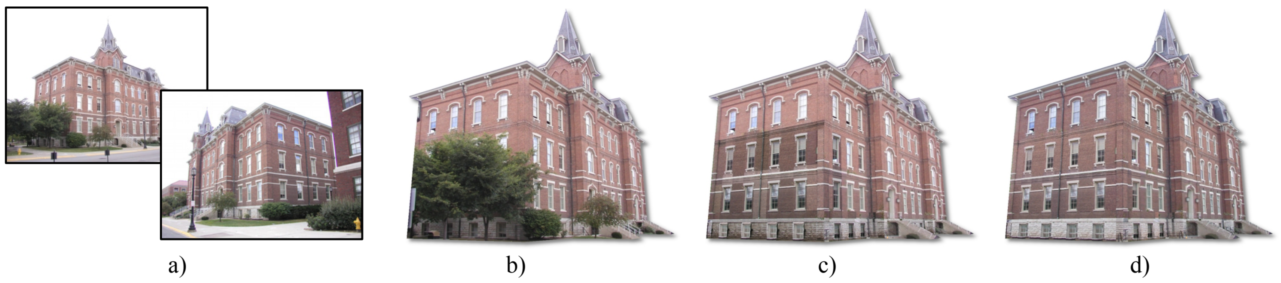


Figure 7. ***Occlusion Removal and Color Equalization****. The system uses the redundant samples of the terminal types to automatically fill occluded surfaces and equalize their color. (a) Two images from a captured model. (b) A view of the recovered model rendered without occlusion removal. (c) The view rendered with occlusion removal (occluded elements in the left image in (a) are automatically replaced with elements from the right image in (a)). (d) The same view but with colors normalized as well.*

Occlusion Removal

Our method uses the production rules and multiple instances of each terminal type to replace parts of the building occluded in the original images. Since in practice it is hard to obtain views of a large structure, such as a building, completely free of occlusions. this provides significant additional flexibilty when capturing images. The result is a full rendering of the captured building, using either projective texture mapping or stylization, despite such views not existing in the original data.

To accomodate occlusion-free rendering, we render an occluded terminal using a selected subset of the image samples. For a given terminal T, we define a function *fitness*$(T, (V_i, T_j))$ to determine the similarity between the terminal and its possible replacements from view V_i using unoccluded terminal T_j of the same feature group. The criteria of the function, in order of importance, are:

- Model Size. T and T_j should be as close in size as possible (determined by their areas of intersection).
- Corner Orientation. T and T_j should have the same corner orientation to reduce imaging artifacts.
- Image Size. A larger image footprint of T_j in V_i is preferred to eliminate resampling artifacts.
- Normal. T and T_j ideally share the same normal in order to match lighting conditions (mostly applicable for projective texture mapping).

Starting with the best available T_j, the terminals' image samples are added to a set of potential replacements for T. Even if no single member of the feature group is sampled from all desired angles, it is often possible to obtain a complete rendering when the entire group is considered. The quality of the final rendering will depend on the similarity between the terminals of the group.

Figure 7 shows occlusion removal and color equalization using projective texture mapping. Figure 7a contains two original photographs. Figure 7b contains a naïve reconstruction where trees and bushes obstructing the building are unwillingly textured onto the model surface. In Figure 7c, the occluding objects have been removed from the surface by using image data from other faces. Figure 7d shows the same rendering with colors equalized and it is more difficult to tell which faces have been replaced.

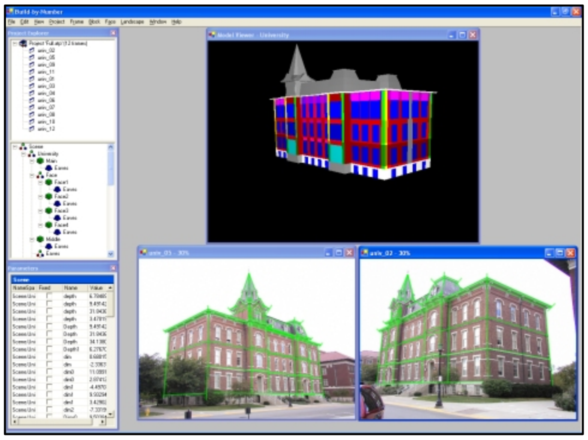


Figure 8. User Interface. *The system allows the user to reconstruct original buildings from images as well as create novel-buildings. The figure shows the modeling window, imaging windows, and tool windows containing a scene graph visualization as well as model parameters.*

5 Implementation Details

Our Build-by-Number system is implemented in C++ on a 3.0 GHz PC equipped with 1GB memory. The user interface is implemented in Windows Forms using Managed C++ (Figure 8). All graphics functionality is implemented in OpenGL. We can render scenes with multiple buildings at real-time rendering rates. Model recovery is performed by minimizing an error function between the edges of the model and user-marked edges as in [5]. We perform the minimization using an implementation of a nonlinear least squares method obtained from the Numerical Recipes in C library [31]. View-dependent texture mapping is implemented using OpenGL's projective texture mapping functionality. Alpha blending is used to weight each texture's contribution appropriately. We use shadow mapping to

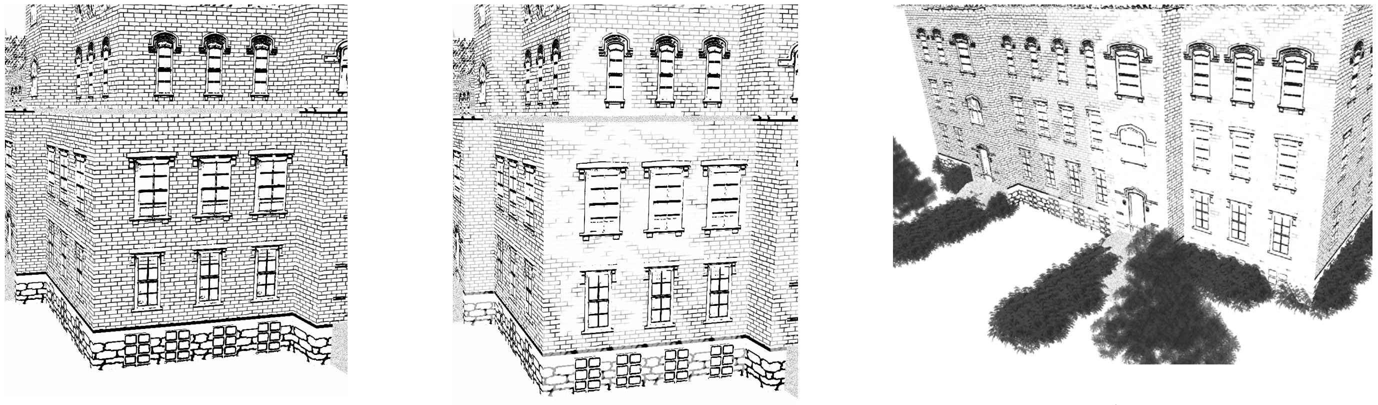

a) b) c)

Figure 9. Stylized Rendering*. Our technique supports procedural stylizations, such as pen-and-ink illustrations. (a) A rendering using the same stylization level for all terminals. (b) The same view but lighter-colored stylizations are used to represent brighter highlighted areas using a diffuse shading model. The light source is near the front-left of the building. (c) Another example rendering with some landscaping.*

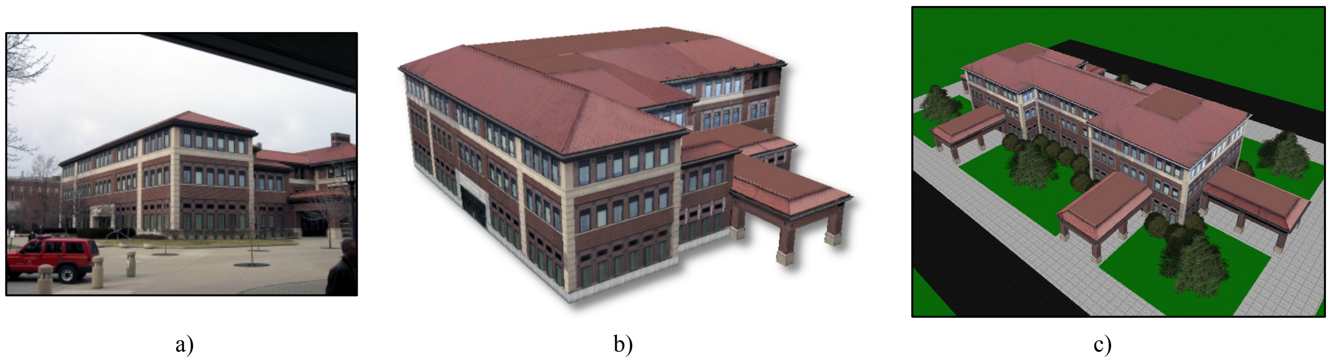


***Figure 10. Projective Texture Mapping**. Our system can also render using color-equalized view-dependent textures. (a) An original image of Administration building. (b) Rendering of original building with occlusions removed and colors equalized. (c) Novel building of the same style together with landscaping.*

prevent the image from being projected onto back-facing and occluded polygons.

6 Results and Examples

With our system we have created several existing and novel buildings based on real-world image data. We acquired datasets varying from 4 to 16 images for seven different buildings: University, Engineering, Music, Administration, Office, Apartment, and Corner. Adding a building to the visualization system is a one time processing effort taking one to two hours to create the model, mark edge correspondences, subdivide the model, mark occluded faces, and stylize the terminals. Once a captured building is available, a novel model can be created and modified on-the-fly using projective texture mapping or stylized rendering.

Using our interactive system, the user can arrange sets of connected building blocks from a pre-defined list of solid primitives and start sketching buildings. An entire building grammar can be applied instantly to a new set of building blocks from a captured building with a single copy-and-paste operation. Alternatively, the user may want to view a captured building, change the size and shape of the original building, or add landscaping. Handles are provided on the building blocks to facilitate their resizing. In the system of this article, the building layouts can be augmented interactively with synthetic ground, texture-mapped sky, trees, and bushes using our landscape painter. The ground plane is divided into small tiles that can be "painted" with grass or cement. Similar to an airbrush, a user can draw a cloud of leaf billboard textures to produce a bush or tree cluster.

Figure 9 shows several pen-and-ink style views rendered interactively by our system. The buildings use the grammar inferred from the University dataset (see Figure 2a for example photographs). Figure 9a and 9b contain close-ups of a novel building created with this grammar. In Figure 9a, the same stylization level is used for all terminals. In Figure 9b, shading is produced using both diffuse shading and our stylized shading model. Lighter-colored stylizations are used to represent brighter areas. The point light source in this example is located slightly below and to the right of the camera. Figure 9c shows a stylized view of the captured University building with bushes added at the base. These renderings give a sketched feel to the scene yet maintain the style of the original structure, require minimal effort by the user, and can be interactively changed and navigated.

Figure 10 demonstrates the use of projective texture mapping to render views of the Administration building. Figure 10a shows one of the original photographs of the building. Figure 10b illustrates the recovered original model free of occluded surfaces and with color intensity equalized. Figure 10c contains a novel building created in the style of the original in about 15 minutes, including the landscaping. Since the new model is more regular than the original, some face production rules were applied individually to maintain vertical coherence. Instead of copy-and-paste of an entire building grammar, our system also affords copying grammars to individual floors and faces.

Figures 11-13 illustrate additional buildings and renderings produced by our system. Figure 11a shows a rendering of the original Office building and Figures 11b-d demonstrate a wide range of modifications to the original building. Fig-

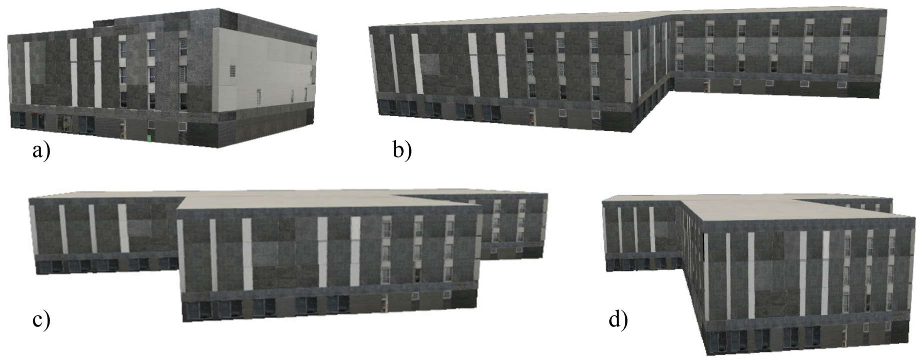


***Figure 11. Office Building**. Our system supports visualizing a wide range of changes to existing buildings. For example, (a) an original recovered model of the Office building, and (b-d) extensions of the building into a L-shaped and two T-shaped configurations.*

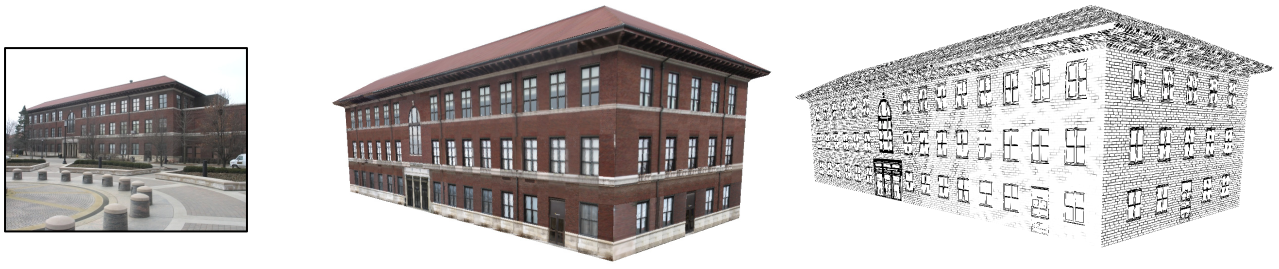


***Figure 12. Music Building**. (a) An original photograph of Music building. (b) Rendering of original model using projective texture mapping. (c) Stylized rendering from similar viewpoint with a light source in front of the near building corner.*

ure 12a contains an example image of the Music building. Figures 12b and 12c show a projective texture-mapped rendering and a stylized rendering, respectively, from approximately the same viewpoint. Figures 13a-c show pictures of the original (a) and extended versions (b-c) of the Apartment building demonstrating how the building might look if additional floors and apartments where added. Figures 13d-e contain a captured photograph and reconstructed model of the Corner building. The large number of occluder objects very close to the ground floor of the building required an aggressive replacement using only the few unoccluded tiles available. Moreover, this building only has two facades; yet using our system we can copy-and-paste the grammar onto the obscured faces of the building configuration yielding a complete building. Figures 13f-g show the adjacently captured photograph and a virtually-reconstructed corner of the building.

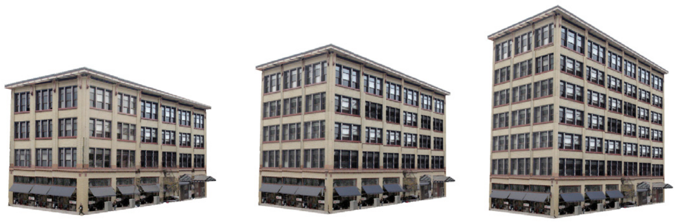
Finally, Figure 14 shows using our approach to generate an in-photograph visualization of building modifications. Figure 14a contains an original photograph. In Figure 14b, we use our system to build a new model. Since during the initial model creation process, our system determined the pose of the camera via the optimization, we can re-project the new model onto one of the original images (Figure 14c) and obtain a glimpse of the modified building in place. While this use of our system does not work for all photographs and does not account for large occlusion changes, it provides a powerful, yet simple to use tool.

7 Conclusions and Future Work

We have presented a method to construct a grammar from photographed and subdivided buildings, enabling the rapid sketching of novel architectural structures in the style of the original. Using several captured models, we show that novel buildings can be designed very quickly and rendered with realism or style comparable to the original structures. We also demonstrated that the extracted procedural rules can be easily adapted to interactive projective texture mapping and to non-photorealistic rendering. Further, our occlusion removal and color equalization algorithms make it possible to capture even highly occluded buildings in varying lighting conditions. The system does not require significant user knowledge and thus is friendly to both non-expert and advanced users.

Our system has several current limitations. We assume the building and its terminals (e.g., windows, doors, etc.) to be static from image to image and for terminals in the same group to be identical. While changes due to diffuse illumination are compensated for, the latter assumption causes some minor artifacts during interactive navigation when windows in a façade are replaced with other similar-style windows yet with different interior content (e.g., shades, curtains, etc.). Furthermore, although we found many buildings conform to our assumed partitioning and available building blocks, it is a not always the case (e.g., the roof-line arch of Figure 13d). One option is to add more fundamental building blocks and/or to break up a building into editable parts conforming to our assumed partitioning.

Looking forward, there are several avenues of future work. First, we are exploring combining our system with an automated city modeling system. Second, we are investigating methods to discover higher-level and more abstract patterns and styles within a building. Third, we are seeking extensions to our approach for generating entire urban spaces in the style of some existing city. We believe inverse procedural modeling to be a very powerful paradigm and



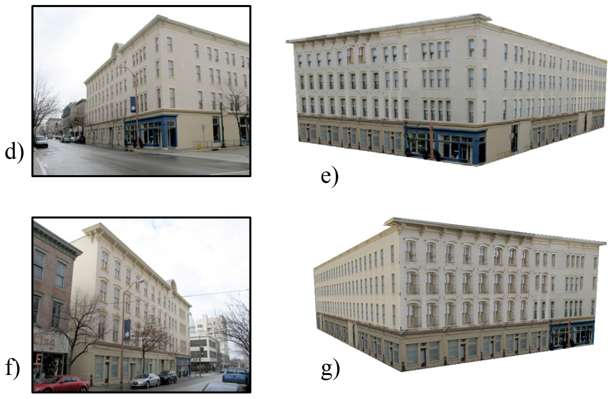


***Figure 13. Apartment and Corner Buildings**. (a-c) Using our system, we can visualize increasing floors/apartments to an existing building. (d-g) We can also generate a complete building from only a corner building by copying and pasting the grammars.*

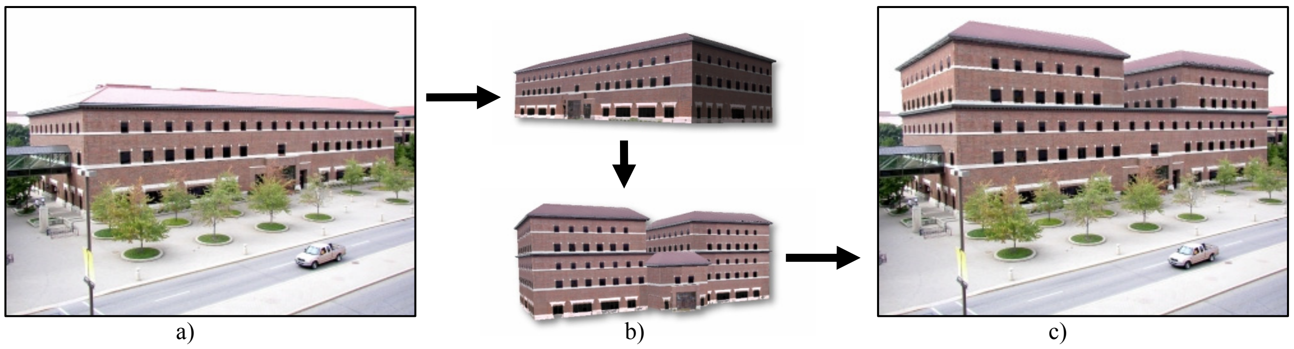


***Figure 14. Engineering Building**. Our system can be used to modify photographs in a perspectively correct manner. This simple, but powerful, ability allows us to quickly visualize new structures in existing scenes. (a) An original photograph. (b) Our method performs the same operations, namely, recover and modify the model. (c) The new model has been projected back into the original image.*

look forward to significant improvements in model generation and visualization in computer graphics.

ACKNOWLEDGMENTS

We are thankful to our colleagues in the Purdue Graphics and Visualization Lab for helping with this project. In addition, this work is sponsored in part by NSF CCF 0434398 and by a Microsoft Research gift -- we are very grateful to both as well.

REFERENCES

[1] P. Prusinkiewicz and A. Lindenmayer, “The Algorithmic Beauty of Plants”, *Springer-Verlag*, 1991.

[2] Y. Parish, P. Muller. “Procedural Modeling of Cities”, *ACM SIGGRAPH*, pp. 301-308, 2001.

[3] P. Wonka, M. Wimmer, F. Sillion, W. Ribarsky. “Instant Architecture”, *ACM SIGGRAPH*, pp. 669-677, 2003.

[4] P. Mueller, P. Wonka, S. Haegler, A. Ulmer, L. van Gool, “Procedural Modeling of Buildings”, *ACM SIGGRAPH*, pp. 614-623, 2006.

[5] P. Debevec, C. J. Taylor, J. Malik, “Modeling and Rendering Architecture from Photographs”, *ACM SIGGRAPH*, pp. 11-20, 1996.

[6] P. Debevec, G. Borshukov, Y. Yu. “Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping”, *In Proceedings of 9th Eurographics Rendering Workshop*, 1998.

[7] MetaCreations, Inc. Canoma. www.canoma.com, 2002.

[8] Eos Systems, Inc. PhotoModeler. www.photomodeler.com, 2005.

[9] RealViz, S.A. ImageModeler. www.realviz.com, 2005.

[10] N. Max and K. Ohsaki, “Rendering Trees from Precomputed Z-Buffer Views”, *Rendering Techniques ’95: Proceedings of the 6th Eurographics Workshop on Rendering*, pp. 45-54, 1995.

[11] L. McMillan and G. Bishop, “Plenoptic Modeling: An Image-Based Rendering System”, *ACM SIGGRAPH*, pp. 39-46, 1995.

[12] M. Levoy and P. Hanrahan, “Light Field Rendering”, *ACM SIGGRAPH*, pp. 31-42, 1996.

[13] Gortler S., Grzeszczuk R., Szeliski R., and Cohen M., “The Lumigraph”, *ACM SIGGRAPH 96*, pp. 43-54, 1996.

[14] D. Aliaga, T. Funkhouser, D. Yanovsky, I. Carlbom, “Sea of Images: a Dense Sampling Approach for Rendering Large Indoor Environments”, *IEEE Computer Graphics & Applications*, 23:6, pp. 22-30, 2003.

[15] D. Aliaga, I. Carlbom, “Plenoptic Stitching: A Scalable Method for Reconstructing Interactive Walkthroughs”, *ACM SIGGRAPH*, pp. 443-450, 2001.

[16] S. Teller, M. Antone, Z. Bodnar, M. Bosse, S. Coorg, M. Jethwa, N. Master, “Calibrated, Registered Images of an Extended Urban Area", *IEEE Computer Vision and Pattern Recognition*, 2001.

[17] A. R. Dick, P.H.S. Torr, and R. Cipolla, “Modeling and Interpretation of Architecture from Several Images”, *International Journal of Computer Vision*, Vol. 30, No. 2, pp. 111-134, 2004.

[18] G. Stiny, “Pictorial and Formal Aspects of Shape and Shape Grammars”, *Birkhauser Verlag*, Basel, 1975.

[19] J. Legakis, J. Dorsey, S. Gortler, “Feature-based Cellular Texturing for Architectural Models”, *ACM SIGGRAPH*, 2001.

[20] I. Shlyakhter, M. Rozenoer, J. Dorsey, S. Teller, “Reconstructing 3D Tree Models from Instrumented Photographs”, *IEEE Computer Graphics & Applications*, 21:3, pp. 53-61, 2001.

[21] R. Zeleznik, K. Herndon, J. Hughes, “SKETCH: an interface for sketching 3D scenes”, *ACM SIGGRAPH*, pp. 163-170, 1996.

[22] A. Shesh, B. Chen, “SMARTPAPER: An Interactive and Intuitive Sketching System”, *Computer Graphics Forum*, Vol. 23, issue 3 (*Eurographics)*, 2004.

[23] J.Y. Oh, W. Stuerzlinger, J. Danahy, “Comparing SESAME and Sketching for Conceptual 3D Design”, *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pp. 81-88, 2005.

[24] Google Sketchup, www.sketchup.com, 2006.

[25] F. Alegre and F. Dellaert, “A Probabilistic Approach to the Semantic Interpretation of Building Facades”, *Georgia Institute of Technology Technical Report*, GIT-GVU-04-31, 2004.

[26] D. Bekins, D. Aliaga, “Build-by-Number: Rearranging the Real World to Visualize Novel Architectural Spaces”, *Proceedings of IEEE Visualization*, pp. 143-150, 2005.

[27] M. Chi, T. Lee, “Stylized and Abstract Painterly Rendering System Using a Multiscale Segmented Sphere Hierarchy”, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 1, pp. 61-72, 2006.

[28] A. Lu, C. Morris, J. Taylor, D. Ebert, C. Hansen, P. Rheingans, M. Hartner, “Illustrative Interactive Stipple Rendering”, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 9, No. 2, pp. 127-138, 2003.

[29] P. Rheingans, D. Ebert, “Volume Illustration: Nonphotorealistic Rendering of Volume Models, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 7, No. 3, pp. 253-264, 2001.

[30] G. Winkenbach, D. Salesin, “Computer-generated Pen-and-Ink Illustrations”, *ACM SIGGRAPH*, pp. 91-100, 1994.

[31] W. Press, S. Teukolsky, W. Vetterling, B. Flannery, "Numerical Recipes in C", *Cambridge University Press*, 2nd edition, 1999.

Daniel G. Aliaga received a B.S. degree in computer science from Brown University in 1991, and a Ph.D. degree in computer science from the University of North Carolina at Chapel Hill, USA in 1999. Since 2003, he is an assistant professor with the Department of Computer Science at Purdue University. He is an active researcher in computer graphics, in particular capturing and rendering large complex environments. Over the years, Dr. Aliaga has developed and published several new algorithms for interactively rendering massive models, recreating complex 3D environments, visibility culling, reconstructing images, estimating camera pose, calibrating cameras, and compressing images. In addition, he has designed complete experimental research systems, in collaboration with Bell Labs, UNC at Chapel Hill, Princeton, and Johns Hopkins University.

Paul A. Rosen received a B.S. degree in computer science from Purdue University West Lafayette, Indiana in 2004. He is currently a Graduate Research Assistant with the Computer Science Department at Purdue University. His research interests lie in the areas of computer graphics, 3D displays, image-based rendering, and 3D scene acquisition and reconstruction.

Daniel R. Bekins received a M.S. degree and B.S. degree in Computer Science from Purdue University West Lafayette, Indiana in 2005 and 2004, respectively. He was a graduater researcher within the Computer Science Department at Purdue University. His research interests lie in the areas of computer graphics, gaming, and mixed reality. Currently, he is employed by Electronic Arts.