Build-by-Number: Rearranging the Real World to Visualize Novel Architectural Spaces

Daniel Bekins* Daniel G. Aliaga†
Department of Computer Science at Purdue University

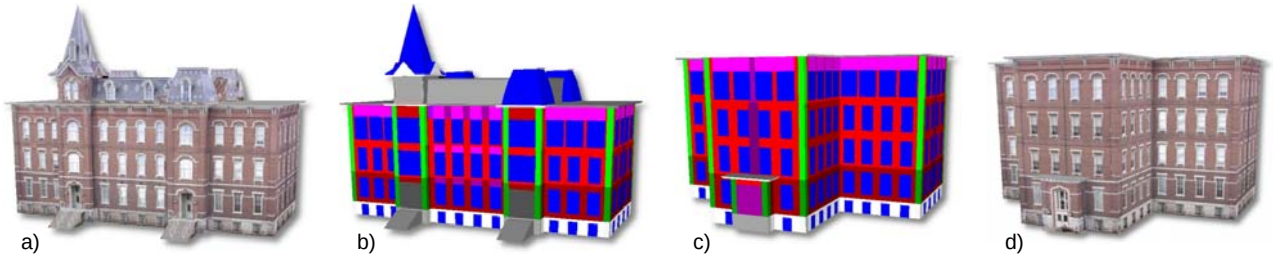

Figure 1. Build-by-Number. A user can reconstruct existing architectural scenes and reuse the acquired data to design and render novel scenes. (a) A rendering of a real-world capture building. (b) The building subdivided into features. (c) A novel model subdivided according to the scheme in b. (d) A rendering of the novel building based on the image data from a.

ABSTRACT

We present Build-by-Number, a technique for quickly designing architectural structures that can be rendered photorealistically at interactive rates. We combine image-based capturing and rendering with procedural modeling techniques to allow the creation of novel structures in the style of real-world structures. Starting with a simple model recovered from a sparse image set, the model is divided into feature regions, such as doorways, windows, and brick. These feature regions essentially comprise a mapping from model space to image space, and can be recombined to texture a novel model. Procedural rules for the growth and reorganization of the model are automatically derived to allow for very fast editing and design. Further, the redundancies marked by the feature labeling can be used to perform automatic occlusion replacement and color equalization in the finished scene, which is rendered using view-dependent texture mapping on standard graphics hardware. Results using four captured scenes show that a great variety of novel structures can be created very quickly once a captured scene is available, and rendered with a degree of realism comparable to the original scene.

CR Categories and Subject Descriptors: I.3 Computer Graphics, I.3.2 Graphics Systems, I.3.5 Computational Geometry and Object Modeling, I.3.6 Methodology and Techniques, I.3.7 Three-Dimensional Graphics and Realism, I.4.8 Scene Analysis.

1 INTRODUCTION

Researchers have achieved impressive results in creating realistic 3D environments as well as reconstructing real-world environments. Computer-aided design (CAD) programs allow precise design and high-quality rendering of three dimensional virtual scenes. More recently, image-based capturing and rendering techniques have made the rendering of real-world 3D environments possible in a more automated fashion. In the case of architecture, procedural modeling approaches have been developed that allow buildings to be quickly generated and rendered based on a set of simple design principles. We seek to merge these paradigms to allow for the fast design of architectural structures that can be rendered realistically at interactive rates.

Each scene creation paradigm mentioned above offers specific advantages, but fails to offer a complete solution to our goal when taken alone. Using only a traditional modeling approach lends great control and flexibility to the designer but requires a high level of expertise and great amount of effort to achieve high-quality renderings. A real-world capture approach offers immediate realism and often requires less expertise on the part of the user, but typically does not offer a convenient way to create novel scene content. Procedural modeling enables very fast design time, but requires the availability of a pre-existing database of scene rules and features. We propose a system that offers the flexibility of traditional modeling, the immediate realism of real-world capture, and the automation of procedural modeling without requiring a high degree of expertise on the part of the user. Such a system can ultimately be employed to quickly generate diverse, high quality content for large urban models based on only a few captured buildings. The urban visualization can then be used for city planning purposes, simulation and training exercises, or interactive entertainment.

Our approach is to use building features taken from real-world capture scenes to create novel architectural scenes (Figure 1). A model recovered from a sparse set of images is subdivided and grouped into feature regions that can be rearranged to texture a novel model in the style of the original. The redundancy found in architecture is used to derive procedural rules describing the organization of the original building, which can then be used to automate the subdivision and texturing of a novel building. This redundancy 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. The novel scene is rendered using view-dependent texture mapping, with a degree of realism comparable to that of the original scene. The complete system is implemented using a standard PC and digital camera, and requires only a moderate degree of modeling knowledge on the part of the user.

* danielbekins@alumni.purdue.edu
† aliaga@cs.purdue.edu

The Build-by-Number system offers the following main contributions:

- A method for quickly designing and editing novel architectural structures based upon image and model data from a captured scene.
- A method of grammar induction from a captured model that can then be used to automatically apply texture to a novel model in the style of the original.
- A method for automatically filling regions of unsampled and occluded data in the image set of an architectural scene.

The remainder of this paper is divided into five sections. Section 2 discusses related work. Section 3 describes the modeling specification used by the Build-by-Number system and the way in which procedural rules for growth and reorganization can be derived. Section 4 discusses the techniques that are used to render a finished model, including the removal of occlusions and a color equalization method. Section 5 covers implementation details of the system. Section 6 discusses results from four captured scenes. Finally, section 7 offers conclusions and ideas for future work.

2 Related Work

The *by-number* concept originates in the 1950's with *Paint-by-Number*, which allowed unskilled hobbyists to create attractive paintings by filling in numbered regions of a pre-made canvas. Hertzmann, et. al., extended the concept to digital imaging with *Texture-by-Number* [1], in which a source image and color coding were combined with a target color coding to produce a high quality image analogous to the source. *Build-by-Number* extends this concept into three dimensions.

Procedural modeling refers to the specification of a model using a set of principles as defined by a grammar. Procedural modeling is most useful for creating models of objects or systems that have a high degree of redundancy or self-similarity. Most notably, *L-systems* have been successful in the modeling of plants [2], and have been used for automatic city and building generation [3]. *Shape grammars* [4], which define rules for the specification and transformation of 2D and 3D shapes, have also been used to model architecture. Wonka, et. al. [5] employ a variation on the shape grammar called a *split grammar* in order to automatically generate architecture from a database of rules and attributes. While these methods provide a means for quickly creating architecture, they do not address the problem of populating the database of building features available for rendering. Parish and Muller [3] define procedural rules for creating basic building features (e.g., brick patterns), but complex decorative features and textures still require manual modeling or painting. While Wonka et al. specifically point out that real-world capture techniques are inherently limited to reconstruction tasks, we propose that a system such as Build-by-Number is well suited for use in generating novel buildings, and requires less expertise than manual modeling or hand texturing to yield good results.

Photogrammetric modeling refers to the process of recovering the dimensions of a 3D model from a set of photographs. While it is a goal in the field of computer vision to recover this information automatically, currently the most robust systems require user input in the form of correspondence data and proxy models. Facade [6] has served as the prototype for several commercial packages [7], [8], [9] and is a good starting point for the Build-by-Number system. In the currently available systems, however, the flexibility made available by the modeling approach does not translate into flexibility in novel design because there is no convenient way to map the image data of a real-world scene to the geometry of a novel scene. Build-by-Number adds the concept of subdivision and labeling to the modeling approach in order to provide such a mapping. This in turn allows a procedural approach to be taken when modifying or creating novel buildings.

Image-based rendering (IBR) is a partner of capture techniques like photogrammetric modeling. Using photographs as data for coloring the objects in a rendered scene is advantageous because a high level of detail and realism is instantly available that would otherwise be difficult to reproduce. *View-dependent texture mapping* (VDTM) [6],[10] is particularly well suited for modern graphics hardware, but there is a major limitation to this method when applied directly to the source images. In many cases, there is an object in the scene covering another object in one or more images. A solution to this *occlusion problem* must somehow approximate the color data for the occluded object in a visually acceptable way.

Current image-based rendering systems use a variety of methods to solve the occlusion problem. For example, occluded pixels can simply be discarded and replaced with unoccluded data from another view. This will fail, however, if the occluding object is close to the surface and therefore occludes it in several views. Unsampled pixels could be filled by interpolating between the surrounding pixels [11], but this is successful only for small regions. Entire faces from one part of the model can be repeated or mirrored onto occluded faces [9], but this assumes an unreasonable degree of redundancy in the scene. Finally, the image can be manually edited to remove occluding objects using 3D image editing tools [12], but skilled manual editing is clearly undesirable. Build-by-Number takes advantage of the subdivision scheme from the modeling phase and the redundant structure inherent in architecture to provide a method for filling in regions of unsampled data. This leads to a much more robust capture and visualization system.

Finally, several commercial packages [13],[14],[15] are currently in use that semi-automatically generate 3D city models using aerial photographs, available geographical information systems (GIS) data, generic texture libraries, and photogrammetrically captured buildings. Since it is too costly to perform a detailed capture for every building in a given area, the majority of buildings must be generated automatically. This leads to a noticeable visual difference between fully captured and generically textured buildings. Our approach offers a sophisticated way to quickly generate novel buildings based on the styles of the surrounding buildings, thus yielding richer visualization.

3 Build-by-Number Modeling

The Build-by-Number modeling system provides a graphical user interface to guide the user through the one-time task of capturing

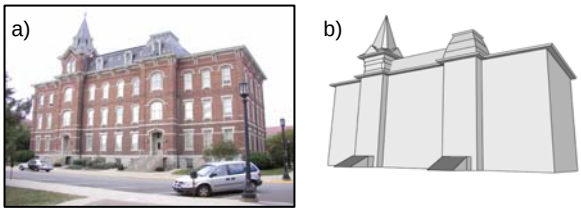

Figure 2. Model Specification. A Build-by-Number model is composed of a collection of geometric solids. (a) One image from a captured image set. (b) A model approximating the geometry of the pictured building.

a new building and then providing the tools to allow very fast creation and editing of novel buildings in the style of the original. Similarly to other photogrammetric modeling packages, the user must take photographs of the desired building, create a coarse geometric model, mark edge correspondences between the model and photos, and mark occluded faces in each image. In addition to these standard tasks, the user must also subdivide the model and place similar features into groups. The system will then automatically recover the model dimensions and camera poses, fill regions of unsampled and occluded data, equalize color and shading between images, derive rules describing building structure, and apply the design of one model onto newly created models.

This section describes the underlying framework of the Build-by-Number modeling system and how it is used to achieve the goals of automation. First, the modeling specification is discussed, including the primitives used and operations that can be applied. This is followed by a description of the automatic grammar induction system and how it is applied to texture the surface of a novel model.

3.1 Model Specification

A Build-by-Number model is a collection of three dimensional geometric solids called building blocks (Figure 2). The blocks are organized into a scene graph describing the spatial relationships between them. 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.).

Each property of the scene graph, i.e. the block dimensions and transforms, is composed of algebraic expressions. These expressions can be simple constants or can combine several free parameters. For example, if the user knows that the building being captured is twice as wide as it is deep, the user can specify the width and depth using only a single free parameter instead of two. This helps to ensure the robustness of the model recovery as well as ensure its accuracy. The model will be recovered by minimizing an error function over the free parameters of the scene (see Implementation Details).

A Build-by-Number model supports two types of operations. An *attachment* operation between two blocks constrains their relative positions such that a specified face from each remains coplanar with the other. Attachment provides the system with block and surface connectivity information that would otherwise be difficult to determine. In addition to attachments, the system supports *subdivision* operations. A block subdivision is the decomposition of a block by a set of planes, resulting in a new collection of smaller blocks called *sub-blocks*. A surface subdivision is the decomposition of a block face by a set of edges into rows and columns, resulting in a grid of smaller faces called *subfaces*.

Figure 3 shows a subdivision and labeling scheme for an example building. Block subdivision is performed to divide the building into floors. Surface subdivision is used to divide the building facade into labeled feature groups representing 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 should be tiled or cropped on a novel face, while the windows and doors should remain the same size.

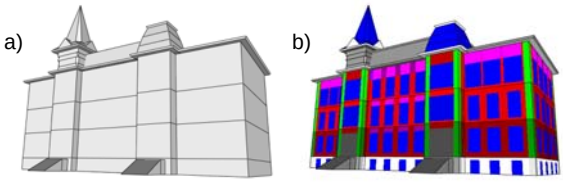

Figure 3. Model Subdivision. The model is subdivided into features. (a) 3D subdivision into floors. (b) 2D subdivision into doors, windows, trim, and brick.

3.2 Design Schemas

A properly subdivided model provides information that can be used to automatically detect patterns in the structure of the captured building. Build-by-Number detects these structural features on several levels of abstraction - the face, the floor, and the entire model - in order to replicate the style of the captured building onto a novel model. Each schema comprises a set of symbolic rules describing the basic ways in which an element can grow or be replicated, along with geometric rules of application.

3.2.1 Face Schema

A *face schema* is a procedural description of a face F, including its symbolic growth rule and its geometric properties. A novel face F' of arbitrary size can be textured with the image data from F by applying this schema. The symbolic rule determines the manner in which features will be replicated, while the geometric information is used to determine precisely how many repetitions of each feature to add to the novel face.

A rewrite rule R for a face F is a grammar production containing symbols that represent individual column subdivisions of F. Identical columns can be combined and marked as repeatable elements with the Kleene star. For example, the face in Figure 4a-b is subdivided into nine columns and can be represented by the string $F = C_1C_2...C_9$, where C_j is the jth colum. The string can then be inspected for identical columns based on its feature labeling. As indicated by color, the pictured face has only three unique types of columns and can be written $F = ABCBCBCBA$. A possible rewrite rule for this face is $R \rightarrow A(BC)^*BA$.

In order to detect identical columns within a face, we define an equality relation between two columns based on the feature labels of their respective subfaces. Let L_{ij} represent the label for the subface F_{ij} at row i and column j of F. If F_{ij} is a terminal face (one with no further subdivisions), then L_{ij} is the user-marked feature label. If F_{ij} is further subdivided, then L_{ij} represents the rewrite rule for F_{ij}. Two rewrite rules are considered equivalent if their respective strings are equivalent. We therefore take two columns $C_a = L_{1a}L_{2a}...L_{ma}$ and $C_b = L_{1b}L_{2b}...L_{mb}$ to be identical if $L_{ia} = L_{ib}$ for all i, $1 \leq i \leq m$.

The column equality relation can be used to derive a rewrite rule for any face. Although there are several ways to combine repeating elements, our method is based on the observation that repetitions of the form AB are typically more visually interesting than those of the form AA. We have also found that combining more than two symbols into a repeating element imposes unnecessary restrictions on the structure of the face. Based on these observations, we devise the following recursive algorithm to determine the rewrite rule for face F:

1. For each subface F_{ij} of F, compute $L_{ij} \leftarrow$ ***rewrite-rule***(F_{ij}).
2. Apply labels to each column $C_j = L_{1j}L_{2j}...L_{mj}$ of F based on the equality relation.
3. Scan the string and mark each reoccuring pair of the form AB (but not AA).
4. Replace adjacent repeated instances of a marked pair with a single instance.
5. Add the Kleene star to all marked pairs.

Some typical derivations are listed below:

1. $ABABABA \Longrightarrow (AB)(AB)(AB)A \Longrightarrow (AB)^*A$
2. $AABAABAA \Longrightarrow A(AB)A(AB)AA \Longrightarrow A(AB)^*A(AB)^*AA$
3. $ACABABA \Longrightarrow AC(AB)(AB)A \Longrightarrow AC(AB)^*A$

The first example represents a very common and straightforward situation. In derivation 2, 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 derivations 1 and 2, we have found this to be counterproductive to correctly texturing a novel face. Suppose we derived the rule $(ABA)^*B(ABA)^*$ for the first example. This actually 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. Also, consider 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 pattern 1 would ruin the vertical coherence between the two floors by adding extra instances of column A to the associated floor of a novel building. Higher-level structure should therefore be imposed by the user through multiple levels of face subdivision.

We now consider applying a growth rule R of $F = C_1C_2...C_m$ to a novel face F' of arbitrary size. We must determine the number of repetitions $k_1, k_2, ..., k_m$ of each column, as well as a scale factor s for those columns of variable width. We desire the scale factor to be as close to *one* as possible. For each non-repeating column C_i, we have $k_i = 1$. We then determine a common multiplier k for all repeating columns such that the remaining width is filled as much as possible without overflowing. By using this common multiplier, we preserve the symmetry and balance of the face structure as much as possible. The remaining width of F' is filled by adding at most one more repetition of each repeating column.

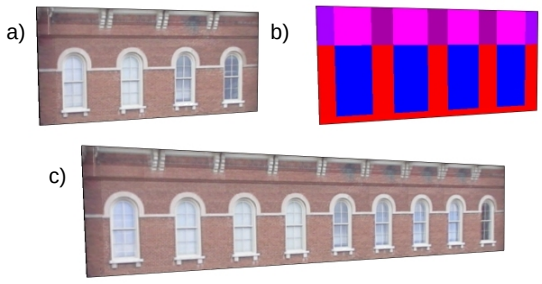

Figure 4. Face Schema. Patterns detected in a face's subdivision scheme can be used to texture a novel face of different size. (a) A face from a captured building. (b) A subdivision scheme for a. (c) The face schema of the original face applied to a larger face.

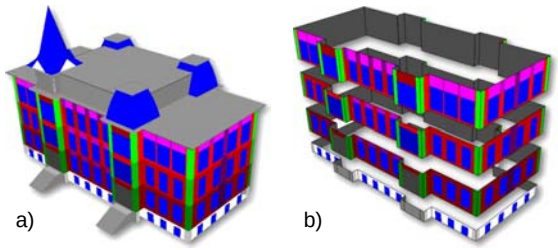

Figure 5. Floor Schemas. A model is composed of several floor surfaces that wrap around the building exterior. (a) The original model. (b) The four floor surfaces of the model. Each floor is composed of horizontally connected faces.

The inclusion or exclusion of each column is determined by searching through the combinations of repeating columns. The combination that yields the scale factor closest to one is chosen.

Once the column multipliers and scale factor are chosen, the texture from F can be tiled onto F' to render the novel face. Note that for subfaces of multiple depth, the texture will be applied recursively to each face. Figure 4c shows the results of the application of a face schema onto a novel face. It can be seen that the novel face rendering remains true to the original and is free of artifacts. This is true in most cases as long as there are no severe shading gradients, cast shadows, or irregular material such as ivy on the surface being replicated. See Section 4 on rendering for more details on how the rendering of novel faces is accomplished.

3.2.2 Floor Schema

A floor schema is a description of all of the faces in a single floor wrapping around an entire model from block to block (Figure 5). This continuous floor surface can be extracted from a model based on the attachment relations between blocks. Each floor of a model is represented as an undirected cyclic graph containing faces as nodes, with each node having a left side and right side edge connecting it to its adjacent faces. The angle between adjacent faces is recorded at the edge connecting them to provide context for the schema application.

Suppose we have a floor schema comprising the face schemas $S = \{F_1, F_2, ..., F_m\}$. We would like to apply this schema to a floor of a novel model comprising the faces $S' = \{F'_1, F'_2, ..., F'_n\}$. To do this, the system must select a schema F_i for each novel face F'_j, then apply the selected schema to the face as described in the previous section.

The system uses three criteria when determining the fitness of a candidate schema for a given novel face. First, the orientation of the face and schema in relation to its adjacent faces is considered. It is frequently the case that the style of a given face is determined by this relationship. For example, an outer corner is likely to have trim or decoration, while an inner corner is less likely to. There are three types of adjacency – inner corner, outer corner, and flat or continuous. Since each face has two adjacencies, this results in nine categories of faces. The best candidate schema will match the novel face in this respect. Second, the difference between the size of a face and the candidate schema are considered. The schema that is closer in size is more likely to be appropriate. Third, if two or more schemas are appropriate in terms of orientation and size, the schema with the least amount of occluded subfaces in the source image set is selected. This will minimize the number of visual artifacts when the schema is applied to the novel face.

Figure 6. Model Schema. An immediate application of the floor and model schemas is to stretch a captured model arbitrarily in any direction (compare with Figure 2).

In our sample models, these three criteria were enough to correctly apply texture to each face of a novel floor (see Section 6).

3.2.3 Model Schema

A model schema is a complete description of all the floors in a model and the connectivity between them. A model schema can be employed to texture an entire model in the style of a captured model in a single cut-and-paste operation. A directed graph contains nodes representing each floor in a model, and edges representing the connectivity between floors. In special cases such as a breezeway or towers, multiple edges can flow into or out of a given floor. Proper floors are separated from base and roof trim by determining the elements of roughly equivalent height that occupy the middle nodes of the graph. The base, bottom floor, top floor, and roof trim are not considered repeating elements, while the middle floors are. Figure 5 shows a model with one base floor and three proper floors connected vertically.

To apply a model schema to a collection of connected blocks, the blocks are first sorted in order of height. The tallest block will be used as the basis block for determining the number of floor repetitions. This is computed by determining the multipliers for each repeating floor that result in the closest match to the basis block height. The block and all its connected blocks are then resized and subdivided according to the portions of the model schema that matches their vertical positions most closely. After subdivision, the floor surface connectivity is updated, and the appropriate floor schemas 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.

Figure 6 shows an immediate application of the model and floors schemas. A captured model can be stretched in arbitrary directions (compare with Figure 2). In the pictured model, only the middle floor is taken as a repeating element.

4 BUILD-BY-NUMBER RENDERING

The Build-by-Number system uses view-dependent projective texture mapping to render both captured and novel structures. This section first discusses the general concept of view-dependent texture mapping. We then discuss how Build-by-Number uses the concept to texture a novel structure from a captured image set, and how this same technique can be used to replace occluded regions of the surface in a captured structure. Finally, we discuss some simple but effective color processing techniques to remove shading from the images before they are used for texture.

1. Transform C into local view space of P' as C'
2. Lookup views and weights $\{(w_1, V_1), (w_2, V_2), (w_3, V_3)\}$ appropriate to C' in view-map of P
3. Set the model transform based on C
4. For each source view V_i
   a. Set texture transform based on V_i
   b. Bind texture image from V_i
   c. Set blending weight w_i
   d. Render texture coordinates from P
   e. Render model coordinates from P'

Figure 7. Novel Polygon Rendering Algorithm. A novel polygon P' is rendered using image data from polygon P.

4.1 Rendering Novel Models

View-dependent texture mapping (VDTM) [10] can be used to render a novel model based on the image data and geometry of a captured model. Standard VDTM can be implemented efficiently by creating a *view-map* data structure for each polygon P of a model. Each source view position is transformed into a coordinate space local to P and projected onto the unit sphere or cylinder in this space. A lookup function determines the three source views closest to the current view during each frame of rendering. These views are weighted according to their proximity to the current view and can then be blended according to this weighting in order to give smooth transitions during animations.

A polygon P' in a novel model can be rendered using data from polygon P. Suppose P' receives its texture from P (as set by a design schema or the user). P' can be rendered from the current view C by using model data from P' and texture data from P (Figure 7). In practice, all of the polygons for a single texture will be rendered before binding a new texture.

4.2 Occlusion-Free Rendering

The view-map of each polygon can be augmented in order to accomodate occlusion-free rendering. Instead of containing only a view position, each entry in the view-map of polygon P will contain a pair (V_i, P_j), where P_j is a polygon belonging to the feature group G of P that is visible in view V_i. This way, all of the texture data in a group can be considered for inclusion in the view-map of any member of the group. Of course it is unwise to add every texture to every view-map in the group. We therefore define a fitness function *view-fitness*$(P, (V_i, P_j))$ to determine the similarity between P and its possible replacements. The criteria of the function, in order of importance, are as follows:

- Equality - If P equals P_j, there is a perfect match.
- Model size - P and P_j should be as close in size as possible (as determined by their areas of intersection).
- Orientation - P and P_j should have the same surface orientation (inner corner, outer corner, coninuous).

1. Let r be the desired sampling density of the view-map
2. Create a list L_P of pairs (V_i, P_j), where $P_j \in G$, and P_j is visible in V_i
3. Sort L_P according to fitness for replacement of P
4. While L_P is not empty
   a. insert the first (best) pair (V_i, P_j) in L_P to M_P
   b. remove all pairs within a radius r of V_i from L_P, except those containing P

Figure 8. View-Map Creation Algorithm. The view-map of polygon P can be constructed by considering each member of the feature group.

Figure 9. Occlusion-Free Rendering. The redundancies marked during the subdivision phase can be used to automatically fill areas of unsampled and occluded data. (a) An initial rendering of a captured building, occluded by trees, shrubs, and other objects. (b) The same building with occlusions removed. (c) Now rendered with color and shading equalization.

- Image size - A larger image footprint of P_j in V_i is preferred to eliminate resampling artifacts.
- Normal - P and P_j ideally share the same normal in order to match lighting conditions (lighting problems will be mitigated through color equalization).

Based on this fitness function, the view-map M_P of polygon P can be constructed by considering each member of the group in order of fitness. The best available pair is added to the view-map until an adequate sampling is reached. Ideally, adding only those pairs that actually contain P will provide adequate coverage (Figure 8).

Even if no single member of the group is sampled from all desired angles, it is often possible to obtain a complete rendering when the whole group is considered. The quality of the final rendering will depend on the similarity between the faces of the group.

4.3 Color and Shading Equalization

It is far more likely that all faces in a group will be similar if some form of color and lighting equalization is performed between them. Given a subdivided model, image equalization is possible by comparing the color data from faces of the same group in different locations on the model. During subdivision, the user marks subfaces that are considered diffuse (usually the surface material of the building, such as brick). The user also marks one or more of the images as color keys that serve as the target for the other images to match. Given this information, the system can perform color equalization between different images, as well as between different surfaces within each image.

We have found that a very simple equalization based on color channel shifting is effective for our image sets. Since we have a subdivided model that is registered with each image, we know the feature group membership and surface normal of each pixel in each image. Using this information, we first determine the average color of each diffuse group from the color key images. For each image, the average color of each diffuse group *for each surface normal* is then computed. By averaging the colors for different surface normals, we can equalize shading between surfaces as well as colors between images. The shift amount for each surface in each image is computed as the difference in the surface average color and the key average color.

Figure 9a shows a rendering of a captured building without occlusion correction. The trees and bushes obstructing the building are textured onto the model surface along with the valid data. In Figure 9b, the occluding objects have been removed from the surface by using image data from other faces. It is possible to see where faces have been taken from surfaces of various shading intensity. Figure 9c shows the same rendering with colors and shading equalized. It is now more difficult to tell which faces have been replaced.

5 Implementation Details

Our Build-by-Number system is implemented in C++ on a 3.0 GHz Dell PC equipped with 1GB memory. The user interface is implemented in Windows Forms using Managed C++. All graphics functionality is implemented in OpenGL. We use an NVidia GeForce FX 5200 graphics card with 128 MB of texture memory. High-resolution images from the digital camera are typically resampled into 1024x1024x24-bit textures and mip-mapped to yield high quality rendering from variable distance. The user can select alternate texture sizes based on the application. Each view also requires a 256x256x16-bit depth map for use in projective texture/shadow mapping. We have had no problem rendering scenes with up to 20 source images using this system.

Model recovery is performed by minimizing an error function between the edges of the model and user-marked edges as in [6]. This method has proven very robust and we have implemented it exactly as described in the original paper. We perform the minimization using an implementation of a nonlinear least squares method obtained from the Numerical Recipes in C library [16]. While the literature makes note of singularities present when performing minimizations involving 3D rotations [17], we found no such problems in practice and therefore do not modify the minimization algorithm.

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 prevent the image from being projected onto back-facing and occluded polygons. Modern graphics cards implement this feature very efficiently, though extra texture memory is needed to hold the shadow depth map. By using shadow mapping, there is a risk of leaving certain areas of the model untextured. To prevent this, we precompute face visibility and render a third pass using faces we know to be fully visible. This pass will not be very expensive as only partially occluded faces need be rendered. We ensure only untextured fragments are updated by basing the blend function on the destination alpha value.

6 Results

We have used the Build-by-Number system to create novel buildings based on image data from four real-world buildings. Statistics regarding the image sets and model composition for each building are listed in Table 1. Each capture takes on the

Table 1. Captured Model Statistics. Each model is made of several blocks divided into subfaces. The model is recovered using wide angle images, while unsampled data can be filled by close-up images.

	Image Set		Model	
Building	Wide	Close-up	Blocks	Subfaces
University	12	4	51	1531
Engineering	7	8	19	1038
Music	4	14	15	1006
Admin	6	4	50	2477

order of one to three hours to create the model, mark edge correspondences, subdivide the model, and mark occluded faces. It should be noted that model recovery is still the largest bottleneck in the capture process – the addition of a subdivision scheme does not add a significant time penalty. Also, it might be possible to significantly improve the capture task, but we consider this to be a separate problem from our method of novel building creation. Once a captured building is available, a novel model can in many cases be created in minutes and textured instantly with the cut-and-paste operation via the model schema. In other cases, the user may want to post-edit the textured model manually. In the case of a pre-existing urban model, Build-by-Number could be used to completely automate the texturing of each building in the model.

The models in Figures 10-13 demonstrate the Build-by-Number process from start to finish. Each figure displays part of an original image set, the reconstructed model with occlusions removed and colors equalized, and an example novel model. All finished models can be viewed in real-time and navigated through interactively. In Figures 10 and 11, the novel models were created in about 5 to 10 minutes each and textured automatically with a single cut-and-paste operation. These models can be further stretched or modified by interactively resizing the blocks, with the texture being automatically updated in a fraction of a second.

Figure 11 demonstrates the use of close-up images to fill in texture data that can not be obtained from wide angle images. This is made possible by the extra edge correspondences made available by the subdivision edges on each block face. The camera pose for close-up images can be quickly obtained with only a few edge correspondences, and take up very little texture memory. Adding many close-ups is therefore not very expensive and can greatly improve the rendering quality of the final model.

The novel model in Figure 12 contained floor configurations that were not present in the original model (such as a double inner corner). The user can still operate quickly and at a suitably high level of abstraction by applying face schemas instead of individually subdividing and applying labels to each face.

7 Conclusions and Future Work

We have presented Build-by-Number, a technique for quickly designing and visualizing realistic architectural structures based on real-world image data. Our results using four captured models show that novel structures can be designed very quickly and are rendered with realism comparable to the original images. It was also demonstrated that procedural growth rules can in many cases be used to automatically texture each novel building in a fraction of a second. Further, our occlusion removal and color equalization algorithms make it possible to capture even highly occluded buildings in varying lighting conditions. All of these are possible without a high degree of modeling knowledge or an understanding of the underlying mechanisms of the system. These results suggest that the Build-by-Number system is a powerful environment visualization tool for both non-expert and advanced users.

The results also make apparent a few limitations to the system. First, the texture tiling mechanism may lead to noticeable seams on the model when viewed closely. Related to this is a limitation on texture memory when very high resolution images are used. We therefore conclude that Build-by-Number is probably not appropriate for producing a single high-resolution structure. Instead, it is more suited for quickly populating a large urban area with buildings that are meant to be viewed from a medium distance during a fly-through or walk-through. This is in line with the intended application of urban visualization, and the realism of the models in this context is very high in our sample captures.

We are already exploring the possible uses of Build-by-Number within a larger GIS-based urban visualization system. We are also interested in applications of the by-number paradigm to the generation and visualization of other types of data such as terrain and city features. These could be used to automatically fill unavailable regions of real-world GIS data, or to create virtual scene data in the style of some existing dataset. As urban datasets increase in detail and availability, the demand for complex and rich visualizations of this data will also increase. We feel that Build-by-Number is a significant contribution toward satisfying this demand.

Reference

[1] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, David H. Salesin. "Image Analogies." *ACM SIGGRAPH,* pp. 327-340, August 2001.

[2] Prusinkiewicz, P., and Lindenmayer, A. "The Algorithmic Beauty of Plants." *Springer-Verlag*, 1991.

[3] Parish, Y. I. H., and Muller, P. "Procedural modeling of cities." In *Proceedings of ACM SIGGRAPH 2001*, ACM Press, E. Fiume, Ed., 301.308, 2001.

[4] Stiny, G. "Pictorial and Formal Aspects of Shape and Shape Grammars." *Birkhauser Verlag*, Basel, 1975.

[5] Peter Wonka, Michael Wimmer, Francois Sillion, William Ribarsky. "Instant Architecture." *ACM SIGGRAPH, pp.* 669-677, July, 2003.

[6] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. "Modeling and Rendering Architecture from Photographs." *ACM SIGGRAPH*, pp. 11-20, August, 1996.

[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] Paul E. Debevec, George Borshukov, and Yizhou Yu. "Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping." *9th Eurographics Rendering Workshop*, Vienna, Austria, June 1998.

[11] William R. Mark, Leonard McMillan, and Gary Bishop. "Post-Rendering 3D Warping." *Symposium on Interactive 3D Graphics,* pp. 7-16, 1997.

[12] Byong Mok Oh, Max Chen, Julie Dorsey, Fredo Durand. "Image-based Modeling and Photo Editing." *ACM SIGGRAPH*, pp. 433-442, August 2001.

[13] Terrain Experts, Inc. Terra Vista. www.terrex.com, 2005.

[14] TerraSim, Inc. TerraTools. www.terrasim.com, 2005.

[15] CyberCity, A.G. CyberCity Modeler. www.cybercity.tv, 2005.

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

[17] Camillo J. Taylor and David J. Kriegman. "Minimization on the Lie Group SO(3) and Related Manifolds." Yale University, 1994.

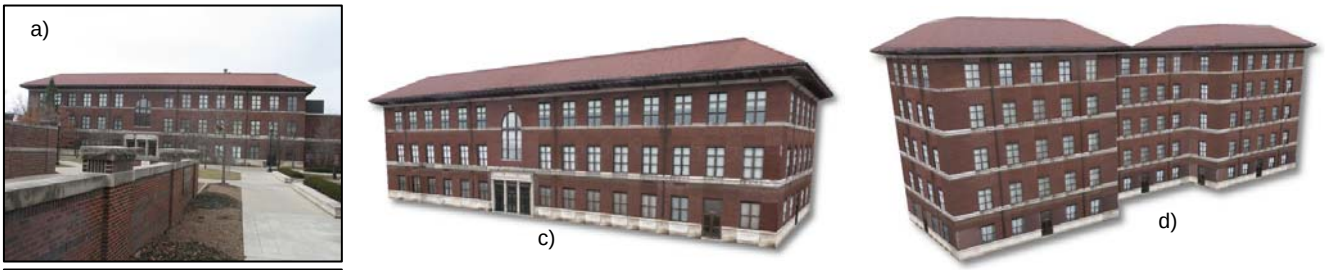

Figure 10. Music Building. The proximity of the trees and other occluding objects to the pictured building makes a straightforward capture impossible. The visible features must be rendered in place of the occluded ones. (a-b) Two images from the original image set. (c) An occlusion-free, color equalized rendering of the captured model. (d) A novel model textured automatically using design schemas.

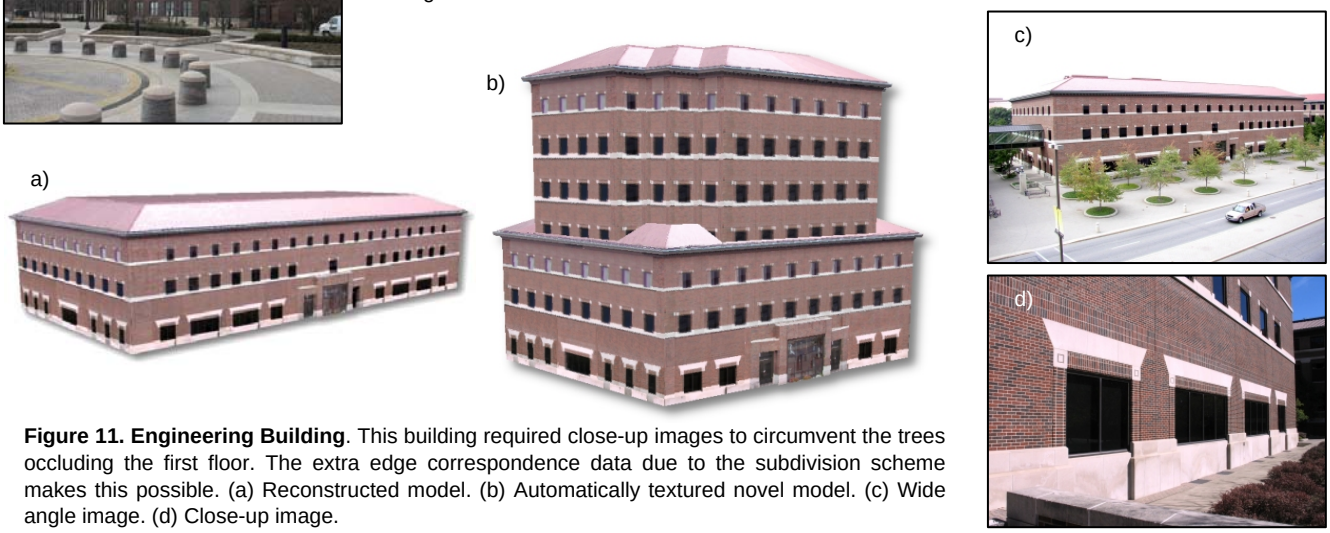

Figure 11. Engineering Building. This building required close-up images to circumvent the trees occluding the first floor. The extra edge correspondence data due to the subdivision scheme makes this possible. (a) Reconstructed model. (b) Automatically textured novel model. (c) Wide angle image. (d) Close-up image.

Figure 12. Administration Building. (a-b) Original images. (c) Occlusion-free, color equalized rendering of the captured building. (d) Full rendering of a novel building based on the image data.

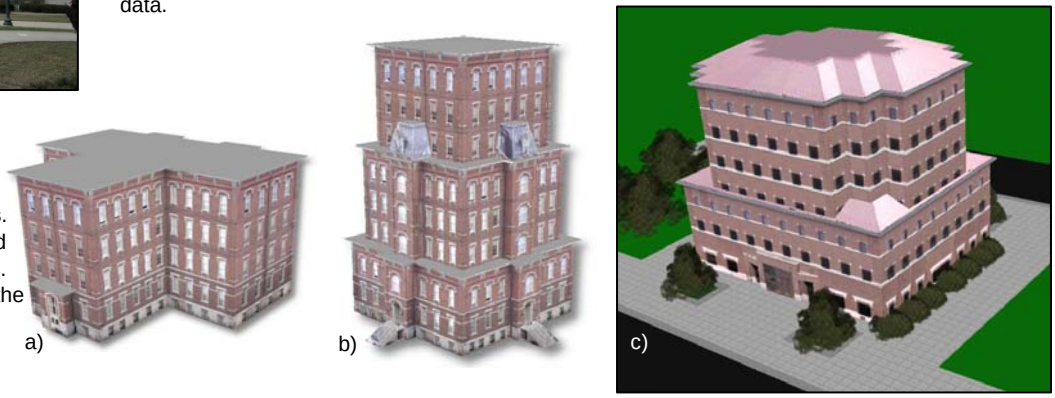

Figure 13. Novel Buildings. (a-b) Novel buildings created based on University building. (c) Full rendering based on the Engineering building.