Fast Architecture Prototyping through 3D Collage

Chuan-Kai Yang and Ching-Yang Tsai
Department of Information Management
National Taiwan University of Science and Technology
No. 43, Sec. 4, Keelung Road
Taipei, 106, Taiwan, ROC
ckyang@cs.ntust.edu.tw,M9609005@mail.ntust.edu.tw

ABSTRACT

In this paper, we propose a new framework for architecture prototyping via the concept of *3D collage*, that is, a combination of geometrically transformed components segmented from multiple source architectures. In short, our proposed framework makes its contribution by featuring three desired functionalities as follows. First, during the construction process, two components can be *snapped* together through the most matched faces. Second, deformation can be performed arbitrarily on any face of the resulting mesh. Third, the assignment of color and texture attributes on the resulting mesh is intuitively and flexibly done in a user-friendly manner. By enjoying a simpler user interface, our system strikes a good balance between efficiency and expressiveness, thus making it particularly appropriate for the purpose of rapid architecture prototyping. Results are shown to demonstrate that our framework is not only good at simulating a wide variety of existing architectures, but also capable of creating even avant-garde architectural styles.

Categories and Subject Descriptors

I.3.4 [**Computer Graphics**]: Graphics Utilities—*Graphics Editors*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Modeling Packages*

Keywords

3D Collage, Automatic Alignment, Mesh Segmentation

1. INTRODUCTION

One of the most important reasons that the field of computer graphics becomes so successful today is the powerfulness of *modeling*, as it plays a crucial role in turning abstract concept into reality. With the help of modern commercial tools, such as *3ds Max* and *Maya*, objects' outlooks, as well their motions, can be vividly constructed or represented. In particular, professional tools such as *AutoCAD* and *Revit* offer complete and sophisticated functionalities for detailedly modeling the exterior, as well as the interior of an architecture. However, the expensive price and huge complexity of these tools have kept them relatively inaccessible to most people. While professional architecture designers may be privileged to make ultimate modeling plans, non-professionals or amateur should also be allowed to share their different perspectives, at least through prototype modeling, to further inspire creativity. Many tools have been developed for this purpose, and according to the type of inputs, there are basically three kinds of approaches: *grammar-based*, *image-based*, and *example-based*. Through the hints of the input, desired model prototypes could be constructed rapidly without starting from scratch, and in some cases even realistic results can be obtained. In spite of their user-friendliness, most such tools are mainly designed for constructing regular structures or mimicking existing architectures, thus implicitly constraining the expressiveness that a modeling tool can achieve.

To address this, this paper proposes a *3D collage* framework that performs desired modeling through a combination of geometrically transformed components segmented from multiple source architectures. Our system offers at least three desired functionalities that render it a simple but extremely intuitive interface for modeling operations. First, during modeling process, when two components, coming from two different models, are held close enough to each other, our system can automatically *snap* them together through the most matched faces. This mechanism greatly saves users from the effort of painstakingly aligning two modeling pieces manually. Second, in addition to supporting affine transformation on each component, a deformation operation is also allowed to arbitrarily and drastically alter the appearance of the resulting composed architecture. Third, as the target model is composed of parts from different sources, this intrinsic segmentation makes the color and texture assignment for the target model relatively easy and intuitive. As a consequence, this *3D collage* system retains the desired convenience seen in aforementioned non-commercial tools, while without sacrificing the freedom or powerfulness that a modeling system is capable of. Most importantly, this simple and intuitive framework serves as a better user interface for stimulating more and further creativity and diversity. To sum up, the contribution of this work can be summarized as the following. First, our 3D collage framework combines parts extracted from multiple sources, thus increasing the overall modeling diversity. Second, our system offers simple and intuitive manipulation functionalities, such as snapping and deformation, which make it particularly suitable for architecture prototyping. As being defined *Artscape* (http://www.torontoartscape.on.ca/about/vision2011/jargon-decoder), *creative process* is "An ongoing, circular and multi-dimensional process of discovery, exploration, selection, combination, refinement and reflection in the creation of something new," where the concept of *combination* aligns pretty well with the very spirit of *collage* employed in this work. One good example of *artistic creations* is shown in Figure 1(b), a photo of a famous model which is

(a) The prototype result generated by our system (b) The original model (photo)

Figure 1: Comparison of our result with the original model.

clearly a novel combination of simple building blocks. Figure 1(a) demonstrates an example result that our system takes a few minutes to construct. Note that here the emphasis is not on how accurate a simulation that our system can achieve, but on how fast, intuitive and user-friendly that our system could be of use for a rapid prototyping.

The rest of this paper is organized as follows. Section 2 reviews literatures related to this work. Section 3 describes the core technique proposed in this work, where we explained the involved preprocessing, supported modeling operations, and implementation details regarding those operations. Section 4 demonstrates the results produced by our system, and makes comparisons if applicable. Section 5 concludes this paper, discusses its limitations, and envisions potential future directions.

2. RELATED WORK

As our work focuses mainly on architecture modeling, we merely review those works more closely related to this. In this regard, there are basically three types of approaches: *grammar-based*, *image-based* and *example-based*. The first type of approaches, i.e., *grammar-based* approaches, are in fact forms of *procedural modelings*, where the task of modeling is often done automatically, procedurally, or recursively, according to some pre-specified rules. In particular, *grammar* is one of the most popular forms of rules used in practice, and its utilization for producing a huge amount of regular architectures is quite common today, such as the works in [4, 24, 8, 25, 13]; among grammar-based works, *shape grammars* are relatively more popular, and some specifically related works can be found from [19, 27, 17, 15]. *Image-based* approaches, the second type of approaches, can also be used for architecture modeling, and they often fulfill the goals through the use of images to derive shapes, depths, textures, etc., such as the works in [5, 18, 28].

The third type, and the most relevant *example-based* approaches, present yet another modeling possibility. In general, based on the provided examples, duplication or modification of the original model can be performed to derive new ones. Paul et al. [20] generalized the idea of *2D texture synthesis* [7, 26, 6] to the 3D world by imposing a 3D lattice structure on the input example model, and the resulting neighboring relationships are used for similarity comparison for consistently synthesizing a larger 3D modeling scene from a smaller one. Merrell et al. [16] proposed a similar approach by analyzing the features of *adjacent boundaries*. A variety of complex shapes bearing similar features can be generated by conforming to the associated adjacent constraints. Funkhouser et al. [10] proposed a *modeling by example* framework, where a desired model is composed of components selected from a 3D model database. Note that here a composing component may not correspond to a model in the 3D database in its entirety but only a part of it, and thus the need for a partial model matching mechanism. Such a framework is simple but intuitive, and as a matter of fact, it inspires the very core part of this work, where we focus particularly on the construction of architectures. Many example-based approaches involve combining two meshes, and in this regard, Sharf et al. [23] improved the original *iterative closest point* [1] (or *ICP* for short) algorithm to become a *soft-ICP* algorithm that allows users to easily and seamlessly combine components from two models in a drag-and-drop fashion. Lin et al. [14] proposed to combine two models by a user-specified transient surface, and the resulting holes are patched through a *localized marching cubes* algorithm. Similarly, Huang et al. [11] and Jin et al. [12] performed model compositions through a *Poisson mesh merging* and a *functional blending*, respectively. Cabral et al. [2] developed a powerful 3D modeling tool for constructing 3D architectural scenes. One of their work's strengths is on its texture preservation, even in the face of deformation or stretching of the underlying 3D models. However, the friendly user interface is in fact at the expense of a non-trivial pre-processing of the input model, as there are numerous constraints on the modeling operations, such as the amount of allowable deformation, and the restricted set of faces that can be used to combine different modeling parts. As a result, the mesh editing flexibility is still affected. Our system, in essence also applies an example-based approach for model construction; however, for the purpose of rapid prototyping,

we restrict the composition surface to be flat to ease users' manual effort and to improve system's performance.

In terms of functionalities and styles, *SketchUp* and *Spore*, developed by the Google and Maxis, respectively, are perhaps by far the most similar tool or system to ours. In *SketchUp*, a user could create not only prototype an architecture, but also fine structures or details associated with the construction should he/she be willing to spend efforts during the process. In fact, *SketchUp* and its successor, *SketchUp Pro*, together with numerous *plug-ins* written by the third parties, have become more and more like a professional and commercial tool. And most importantly, more complete functionalities inevitably make it more complex, at least for novice users. By providing only core and necessary functionalities, our system presents a more friendly and intuitive interface, thus making it a better choice for a rapid architecture prototyping. In *Spore*, users can create all kinds of species and control their development. There are multiple stages in the game, and during the *tribal stage* and *civilized stage*, a *building editor* is provided for constructing a simple architecture. The construction starts from a fixed base, and on top of which one or two more levels of supported building blocks can be added, together with doors, windows, ornaments, and textures to be placed on the surface of the resulting architecture. There are, however, several differences between *Spore* and our system. First, instead of a two or three story building, our system offers much more modeling possibility and diversity. Second, in terms of vividness, our system has the great advantage of using components extracted from realistic 3D models. Finally, as far as deformation is concerned, the simple *linear morphing* mechanism provided by spore is less flexible than our approach of using a *Bezier patch*, to be discussed later. As a last remark, we adopt the same texture assignment operation as used in *Spore* for its simplicity and intuitiveness.

3. MODELING THROUGH 3D COLLAGE

As grammar-based approaches are less intuitive and image-based approaches are less expressive, we resort to an example-based approach, which in principle puts together the functionalities of *Spore* and *modeling by example* [10].

3.1 System Overview

Figure 2 demonstrates the system overview of this work. We first collect 3D models from the Internet or construct one by ourselves, and all the models are converted into polygonal meshes if they were not to ease the ensuing processing. These 3D models are then pre-processed as follows. First, a model is segmented into multiple components to be used as building blocks in the ensuing 3D collage process. Second, for each of these segmented components, some of its important *meta data* are extracted as well for supporting its real-time manipulations in the following model construction phase. Each such component and its meta data are then added into our database. At run time, there are two operation modes, *normal mode* and *texture mode*. In the normal mode, we concentrate on shaping our desired architecture, while in the texture mode, we focus on setting up the colors and textures for the resulting model. In addition, in the normal mode, a friendly user interface is provided for selecting desired components, and when two components are held close, our system can perform automatic alignment; that is, these two modeling pieces can be merged through the most matched face. Moreover, a flexible surface deformation mechanism is also provided to further help users to construct their wanted shapes. We now describe each of these processes and operations in details.

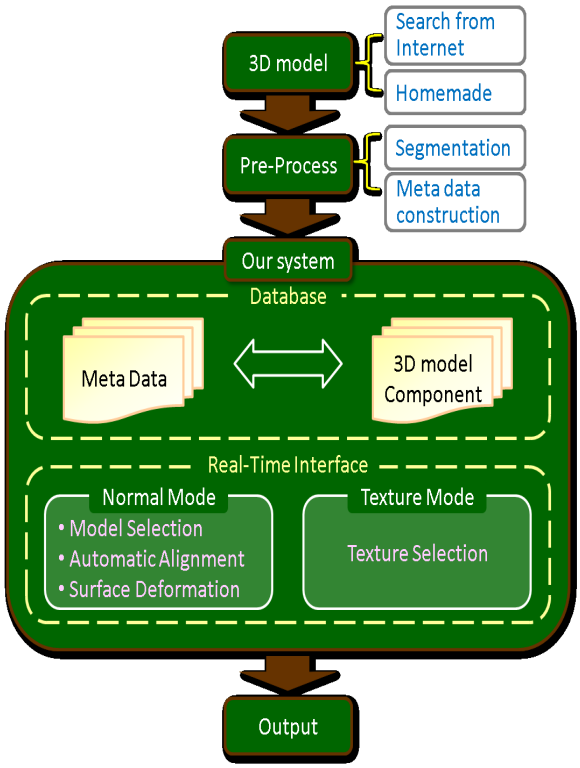
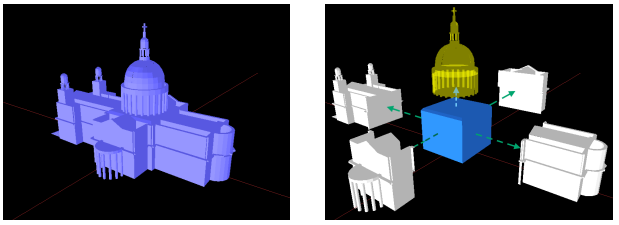


Figure 2: System overview of this work.



(a) The original model (b) The segmented components

Figure 3: An example of model segmentation. (a) The original model. (b) The segmented components where different colors correspond to different labels.

3.2 Preprocessing

As mentioned previously, the goal of model pre-processing is to enrich our model/component database, and to support later real-time modeling operations. As a result, there are correspondingly two tasks to perform at the pre-processing stage, i.e., model segmentation and meta data construction.

3.2.1 Model Segmentation

Figure 3 shows an example of model segmentation, where we divide a given 3D model into six components. For convenience, depending on a component's relative position within the original model, we classify and label each component as one of the following four types: *central*, *top*, *bottom* and *side*. For example, we mark the central, the top and side components in blue, yellow and white, respectively, in this figure to represent their different types. Such a labeling or classification is helpful in later component collage process as it is more intuitive to combine a top component

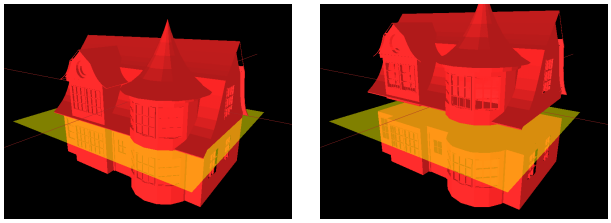


Figure 4: The segmentation process in our system.

N_0: Vertex Indices of F_0, Vertex Indices of F_1, ...
N_1: Vertex Indices of F_0, Vertex Indices of F_1, ...
...

Figure 5: The format for reference meta data.

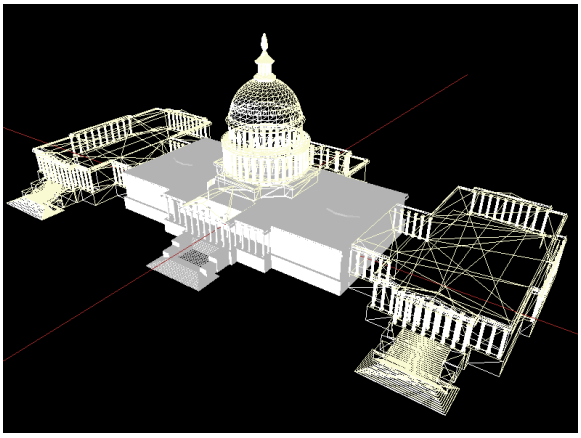


Figure 6: An example of model selection.

with a bottom component than to merge two top or two bottom components from two different models. Figure 4 further demonstrates the segmentation process, where we slice a model to be segmented, as marked in red, by a size-adjustable plane, as marked in yellow in this figure. Note that the intersection of the model with the plane is calculated to identify the intersecting polygons on the original model, and then the portion containing these intersecting polygons is *re-meshed*, so that the slicing could form two separate and complete meshes. In addition, the resulting holes or the borders on both models due to this segmentation are also automatically patched with polygons to facilitate potential future alignment operations to be discussed shortly.

3.2.2 Meta Data Construction

In our current implementation, there are two types of meta data to collect for each segmented component, *normal meta data* and *reference meta data*. As our 3D models are from different sources, it is not always the case that each face is associated with its proper normal (vector) information. Therefore we first compute the normal for each face if its associated normal information is missing. Next we calculate the total number of normals in a component to complete the normal meta data construction. Note that this number must be finite as the number of faces for a model is finite. Regarding the reference meta data, its format is shown in Figure 5, where N_i denotes the i-th normal vector, and F_i denotes the i-th face having this normal. Basically this table records the faces and their composing vertices for each normal, so that at run time when a normal given, we could efficiently locate all faces/vertices whose normals align with this normal. All the meta data information, together with the corresponding modeling components, are added into the database for later use.

3.3 Modeling Operations

Once the pre-processing is done, we are ready for the next stage, i.e., modeling by 3D collage. To ease the modeling process, our system offers several intuitive and user-friendly operations, which are now described in turn.

3.3.1 Model Selection

Based on four types of components mentioned in Section 3.2.1, we could perform model selection as follows. Due to space limit on the user interface, each time a group of *five* components of the same type are shown simultaneously, and users can scroll to the previous or next group to browse another five components. When a component is chosen for a preview, on a side window the original model containing the selected component is drawn for reference, as shown in Figure 6. Note that the selected component is drawn in solid color, while the rest of the original model in a wire-frame mode. We believe that with the original configuration available for reference, it will be more comprehensible for composing the selected component into a desired new design. Also notice that for the sake of efficiency, when previewing a component, only the model, but not its associated meta data, is loaded. Only when a component is explicitly requested to be loaded into the main composition window, its relevant meta data will then be loaded into the memory, to enable ensuing real-time modeling operations.

3.3.2 Automatic Alignment

As each component, as well its meta data can be freely loaded, we next describe how our system combines multiple components together to form the final architecture. First of all, our system surely supports a component's *affine transformations*, such as *translation*, *scaling* and *rotation*. Note that the *shearing* operation, though an affine transform, is currently not provided as it is rarely used. In addition, our system features a most desired functionality: *automatic alignment*, which allows two components to be combined or aligned in a natural way. Figure 7 demonstrates one such example where two components seamlessly and harmoniously merged into one. One should pay more attention to how the upper component scales up to match the lower one. Note that here we assume the corresponding faces of the two components before merging are *flat* so that the ensuing alignment process could be greatly simplified. In the case when the merging surface is not flat, one could circumvent the flatness restriction by first segmenting or deforming the corresponding faces to be aligned. The involved segmentation and deformation operations have been described or will be discussed shortly. Furthermore, since there exist many ways for merging two components, to avoid possible ambiguity, we treat one of the components as the *master*, whose geometric properties such as position, size and orientation all stay put throughout combination process. The other component is thus called the *slave*, which adjusts itself to match the master. In terms of this "master-slave" relationship, the upper component in Figure 7 is the slave while the lower one the master.

The aforementioned master-slave relationship can also be extended to more than two components, and the roles of master and slave are also interchangeable. Figure 8 demonstrates how we use the

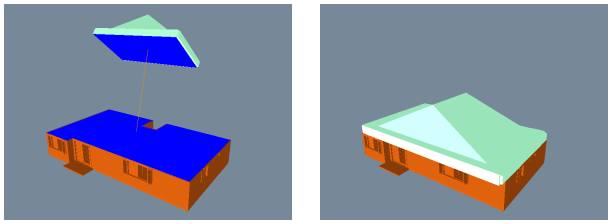


Figure 7: An example for combining two models.

master-slave structure to build a modeling hierarchy. Note that to save space, all images are shown in small sizes. However, they are in fact stored with a higher resolution, and therefore can be enlarged for a better observation. In Figure 8(a), the first component is loaded. In Figure 8(b), the second component is loaded. By default, our system always set the newly loaded model to be the new master. In Figure 8(c), by re-assigning master to be the first component we use it to load another slave component. Similar operations can be carried on, until all desired components are loaded, as shown in Figure 8(e), an enlarged version of the same configuration. As can be seen from all these figures, there always exists a line between each master-slave pair. In our current implementation, this line links the *centroids* of the two components, where the centroid of a model is obtained by averaging all its composing vertices. Once the connecting line is determined, the *attachable faces* on the master and slave components can be determined as well, and later on we will show how these faces can be used for merging the slave component onto the master component. As shown in Figure 8(e), each connecting line must intersect with both its master component, as well as its slave component. Starting from the intersecting locations on both components, we could identify the corresponding polygons where the intersections reside in. By treating these polygons as the initial attachable areas, we gradually grow these areas towards edge-adjacent polygons as long as they all share the same normal vector. This *flooding-like process* eventually would lead to our desired attachable faces on both components. Note that the meta data we collected during at the pre-processing stage are used to significantly speed up this area growing process.

With the attachable faces determined, we now are ready to describe how the alignment or *snapping* operation, shown previously in Figure 7, is performed. There are basically two phases during the alignment process. Figure 9 demonstrates the first alignment phase, where three steps are involved. First, we calculate the normals on both attachable faces, denoted as N_a and N_b in this figure. Second, as the goal is to align two attachable faces, it is therefore equivalent to align N_a with $-N_b$, which can be simply done through a rotation. The rotational axis can be derived by simply calculating the *cross product* of N_a and $-N_b$, and the rotation angle can be computed as well either by the *inner product* or by the *cross product* of N_a and $-N_b$. Finally, after rotating the slave component to make both attachable faces parallel to each other, we calculate the centroids of these two attachable faces, and translate the slave component so that its centroid co-locates with that of the master component.

As performing the first alignment alone only aligns the normals of two faces, we thereby further aligning the directions of edges on both faces to achieve an even better alignment, as shown in Figure 10. Currently this is done by analyzing the histograms of edge direction vectors on both faces for matching the most frequent edge

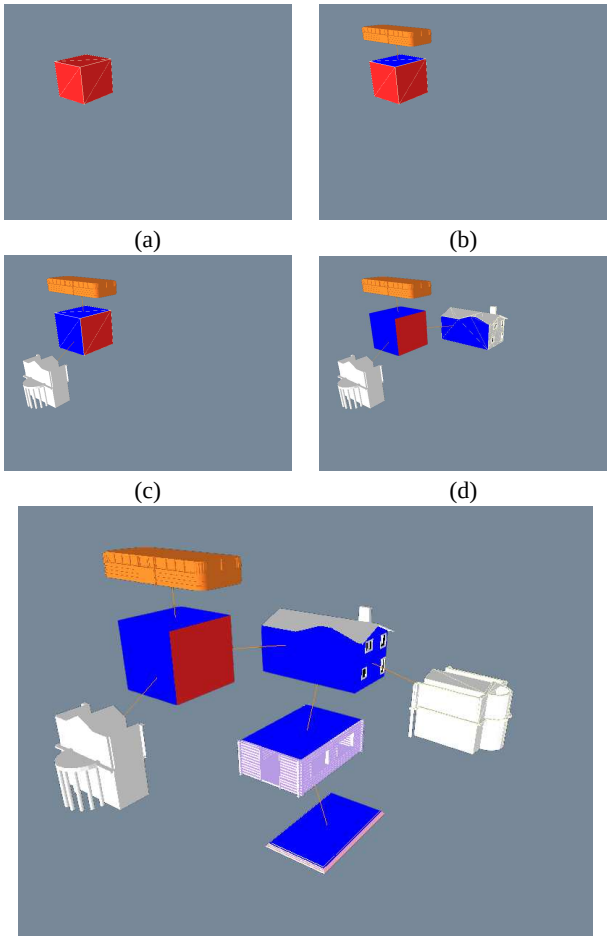


Figure 8: An example for loading multiple components.

direction vectors of the slave's face with that of the master's face by a simple rotation. A more robust implementation could make use of the *principle component analysis*, or *PCA* for short, to align the edge directions trend on both faces.

3.3.3 Surface Deformation

As mentioned previously, in addition to providing affine transformations for the manipulation of components, we further add the functionality of surface deformation to enhance the expressiveness and flexibility of our system. There are many possible schemes that can be adopted for surface deformation, such as *Bezier surface*, *B-Spline surface* [3], *NURBS surface* [21], and *free form deformation* or *FFD* for short [22], and so on. Although the latter three offer more expressiveness through a finer manipulation of the associated *control points*, their more involved operations nevertheless make them relatively less suitable for the purpose of rapid architecture prototyping. For the sake of simplicity, our current implementation adopts the *Bezier surface* representation for the deformation purpose (see [9] for a more detailed introduction), and thus makes two assumption about the face to be deformed. First, the face should be four-sided or can be made so by inserting new or removing old vertices. Note that many small adjacent faces sharing the same normal can be treated as one for the deformation purpose. Second, we assume the deformation to be *local*, that is, it should be contained within the face. As a result, the edges on the face stay fixed to main-

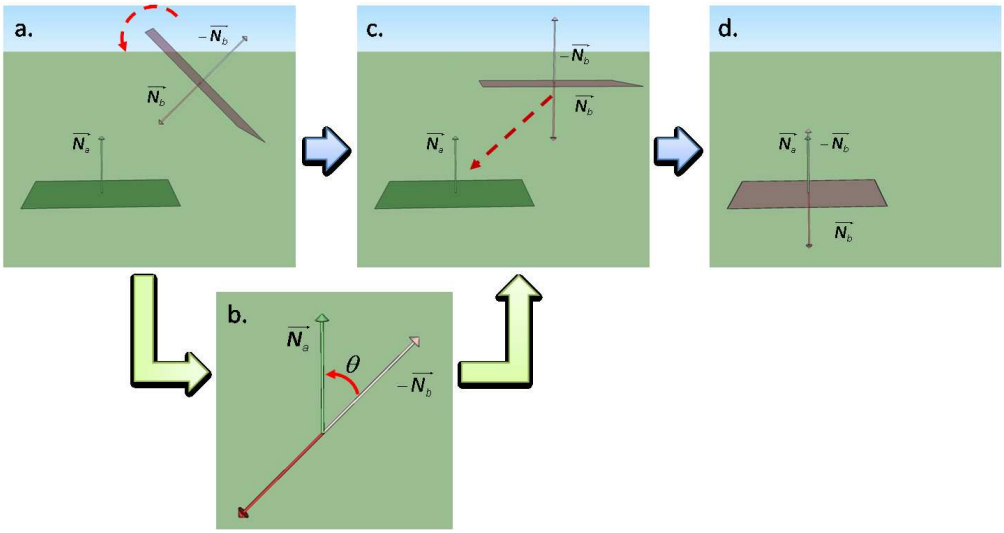


Figure 9: The first alignment phase.

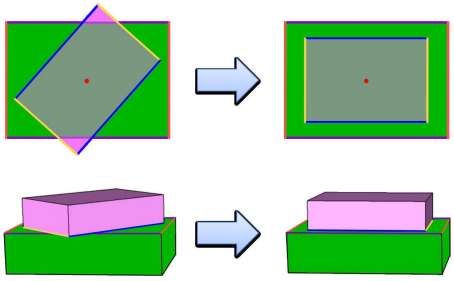
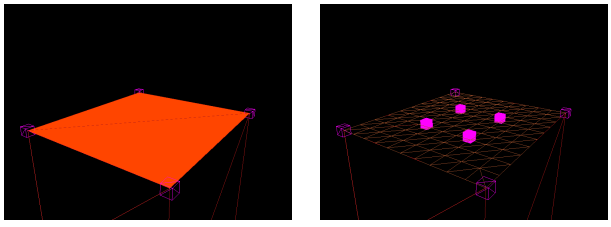


Figure 10: The second alignment phase.



(a) (b)

Figure 11: The initial Bezier patch. (a) The face to be deformed, marked in orange, and the four corner control points, marked in purple wire-framed cubes, on the initial fitted Bezier patch. (b) The linearly interpolated central four control points, marked in purple solid cubes, and the resulting refined mesh.

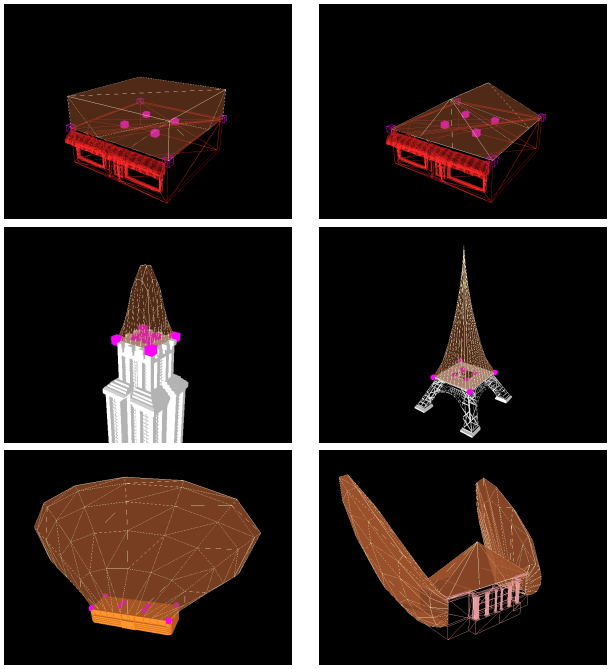


Figure 12: Examples for our versatile surface deformations, where the deformed parts are marked in orange.

tain the locality, and thus avoid influencing adjacent faces. Based on these assumption, we first fit an initial Bezier patch on the face to be deformed, as shown in Figure 11, and central four points can be linearly interpolated, and used to deform the face to a desired shape. To make the deformed portion look smooth, we also allow users to dynamically change the *mesh resolution*, as being demonstrated by the example of a 10×10 mesh resolution in Figure 11(b). Note that here the mesh is uniformly partitioned. One may also adopt an *adaptive partitioning* scheme that is *curvature-dependent* to strike a balance between quality and storage. However, we opt for a simpler scheme as such a consideration is not deemed very critical.

We argue that aforementioned assumptions do not seem as restricted as they look. Figure 12 demonstrates the variety of examples that our simple deformation scheme can generate, and most importantly, in an extremely easy manner. Note that our system also offers the *grouping* and *un-grouping* mechanism that allows users to selectively group the central four points shown in Figure 11(b) to further ease the related manipulation.

3.3.4 Texture Assignment

To make a prototyped architecture look more pleasing, it is often necessary to apply textures on the resulting model, and an example can be found from Figure 13(a) and Figure 13(f), where the second one is a texture-mapped result of the first. Our textures come from either the Internet or accessory data of collected 3D models, and therefore at the pre-processing stage not only model segmentations, but also texture extraction are performed. One subtle issue concerning texture mapping is the following. Oftentimes instead of applying a single texture to the entire model, a more desired scenario is to selectively assign textures to different partitions of a model. As our modeling strategy is through 3D collage, we already have recorded the information regarding each composing component, thus making it very natural to support texture mapping not only on the entire model, but also on each individual component. Note that, for every model that we have collected, it is at most with two or three textures. Furthermore, adjacent *coplanar* faces are associated with the same texture, thus making the ensuing texture assignment process relatively easier. However, we must point out that in the case each face has its own individual texture, the involved texture assignment process would become unavoidably tedious. Figure 13 demonstrates our system's flexibility by a sequence of texture assignment process on different parts of the composed model to arrive the final configuration.

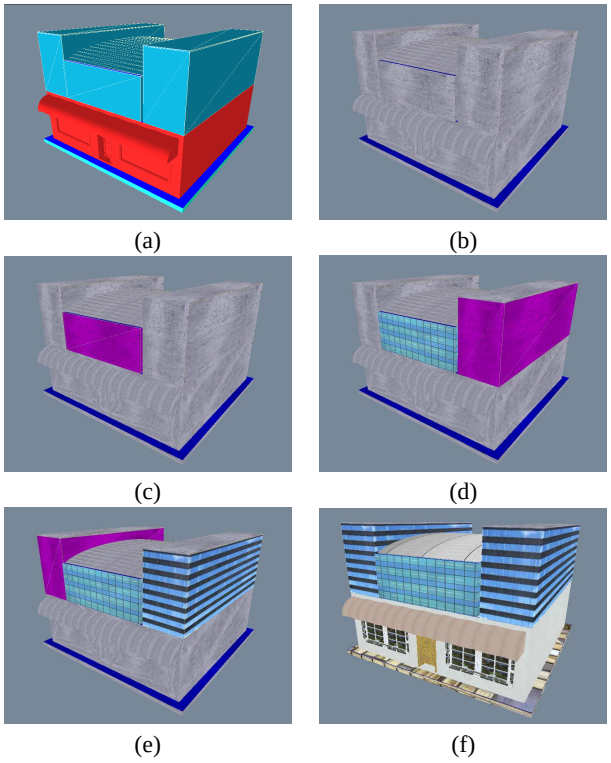


Figure 13: An example for texture assignment. (a) A prototyped model in the normal mode, where components coming from the same source model are marked with the same color. (b) The same mode but viewed in the texture mode, where the default system texture is *cement*. (c)-(e) Texture assignment process, where the portion to be textured is first marked in purple. (f) The final textured model.

4. RESULTS

Our experiments are conducted on a machine with an Intel Core 2 2.83GHz CPU and 2GByte memory, running on MS Windows XP. The programming language for development is Visual C#, with DirectX 9.0c. Table 1 summarizes some pre-processing statistics. As can be seen from this table, our pre-processing is moderately fast. On the other hand, it is also efficient enough to support desired real-time alignment operations.

To prove the expressiveness and efficiency of our proposed framework, we try to mimic some existing architectures via the functionalities provided by our system and the results are shown in Figure 14. The original models (photos) are also shown for the purpose of comparisons. The prototype construction times for these models range from 5 to 20 minutes, while most of them are within 10 minutes. The numbers of involved components are from 3 to 11, and most of them are below 10. In addition, three of them make use of Bezier patches for surface deformation. We should point out that some of the noticeable differences in these images are in fact due to the lack of proper textures. Furthermore, as many models came without textures or without proper texture coordinate information, our automatic and global texture mapping may sometimes cause undesired artifacts as observed. In spite of these imperfections, in the sense of rapid architecture prototyping, we believe these results have still clearly justified the usefulness of our system.

To further demonstrate what our system is capable of, even under such a simple framework, we generate a weird-looking model within 30 minutes, and a model of city within 1 hour, both are shown in Figure 15. Although when compared with [16], a city's construction time for which is less than half an hour, ours is relatively time-consuming, this example nevertheless shows the great modeling potential of our system. One should also pay attention to the fact that, the possibly longer processing time is due to the more involved manual efforts, which at the same time could lead to more versatile building styles than the ones created by an *example-based synthesis* framework [16]. A more vivid demonstration of our system can be found at http://www.cs.ntust.edu.tw/ckyang/modeling.avi.

Finally, we compare *SketchUp Pro*, *Spore* and our system, in terms of functionalities or complexity, as listed in Table 2. Note that for *SketchUp Pro*, we refer to the newer version of *SketchUp*, and it does not include any extra *plug-ins* written by the third parties. First, we compare the functionality of *segmentation*. In *SketchUp Pro*, segmentation is not directly supported. However, such a functionality can be achieved by making a copy of the original model, and using *set difference operations* to delete the unwanted parts from the original model and the copied model to obtain the desired segmentation. Though this approach offers more segmentation flexibility of segmentation, as the segmentation surface needs not to be a plane, it is at the expense of more troublesome operations. For example, the deletion of both parts have to be *complementary* to each other so that an equivalent segmentation can be made. In *Spore*, currently the segmentation functionality is not offered. On the other hand, such a functionality is provided in our

Table 1: Some pre-processing statistics information.

Model	# of vertices	# of faces	preproc. time
Bower	662	712	78 ms
Supermarket	1,451	2,666	1,000 ms
Tower	5,235	9,351	5,500 ms
House	4,852	9,195	6,218 ms
Arc de Triomphe	6,742	12,684	7,484 ms
Empire state building	18,704	16,194	21,843 ms
La Tour Eiffel	24,912	12,456	22,562 ms
Church	10,365	26,821	64,328 ms
Colosseum	15,538	30,000	172,109 ms

(a) (b) (c) (d)

Figure 14: (a) and (c) are the results generated from our system, while (b) and (d) are the original models that we want to mimic.

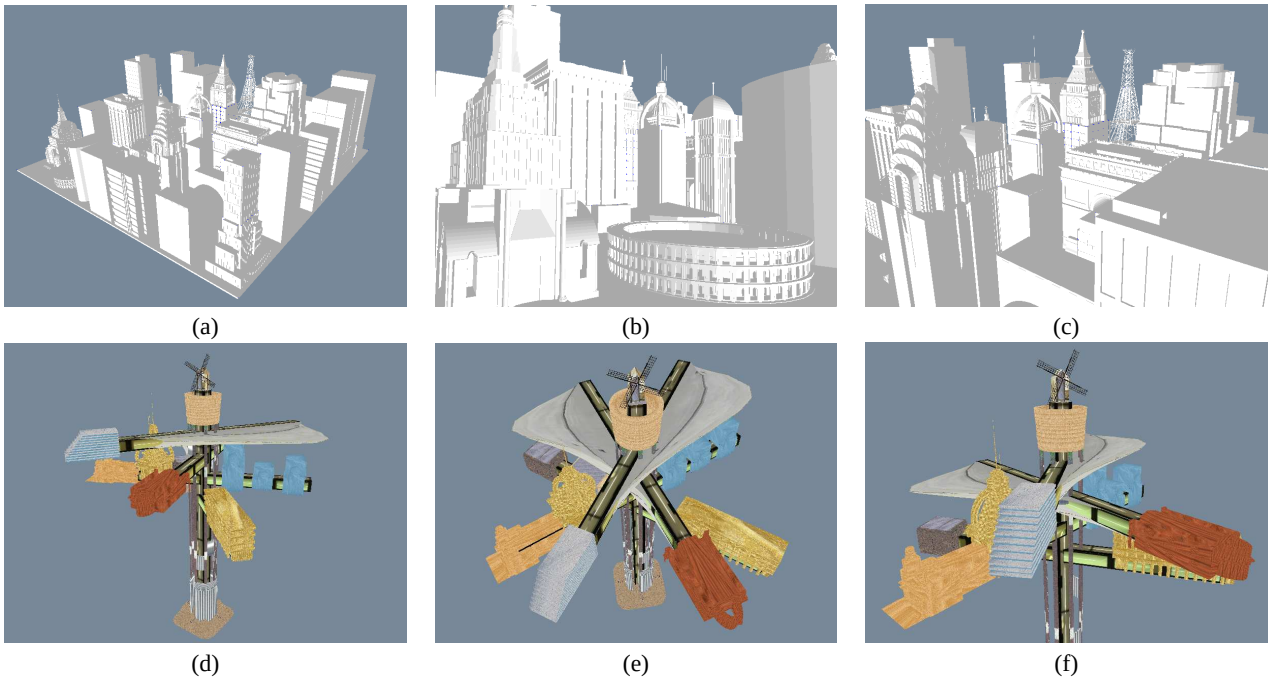


Figure 15: Two examples generated by our system. (a)-(c) are different views for a city model, while (d)-(f) are different views for a weird-looking model.

Table 2: Comparisons with Google SketchUp Pro and Spore.

Functionalities/Complexity	SketchUp Pro	Spore	Ours
Segmentation	Indirectly	No	Yes
Texture Assignment	Polygon-based	Model-based	Model-based
Alignment	Manual	Automatic	Automatic
Deformation	No	Yes	Yes
Result Complexity	Various	Fixed	Various
Operational Complexity	High	Medium	Low

system, and it also serves as an important functionality to extract useful and re-usable components from existing models. Second, we compare the *texture assignment* functionality. In *SketchUp Pro*, it is a *polygon-based* operation. While offering maximal flexibility, such an approach may become overwhelming in a rapid prototyping scenario. *Spore* and our system, on the contrary, apply a model-based approach to ease this potentially tedious task. The third functionality to compare is *alignment*. *SketchUp Pro* does not offer this mechanism, while *Spore* and our system adopt it to further simplify the prototyping process. Note that, as shown in Section 3.3.2 and Figure 7, our system provides an additional automatic resizing capability, which can be turned off optionally, to allow an even more smooth model composition process. *Deformation* is the fourth functionality for comparison. Currently *SketchUp Pro* does not allow this, unless some *plug-ins* are installed. As mentioned in the Related Work section, both *Spore* and our system permit users to deform the resulting model, although through different mechanisms. Next, we compare our system with *SketchUp Pro* and *Spore* on *result complexity* and *operational complexity*, respectively. By *result complexity*, we refer to the achievable resulting model complexity when a particular system is used. In *SketchUp Pro*, the result complexity can vary dramatically, and it basically depends on the skillfulness of users. In our system, such complexity can also ranges widely, but it mainly depends on the complexity of component models. In *Spore*, the result complexity is relatively fixed, as the involved building components are fixed and limited. Finally, speaking of *operational complexity*, we meant to compare the time that it takes for a novice user to learn to master the manipulation of a system. In this regard, *SketchUp Pro*, a very friendly tool for building simple shapes from scratch, aims at gradually becoming a powerful modeling tool, and at the same time evolving into a user interface that is more and more complicated and sophisticated. *Spore* tries to strike a balance between convenience and flexibility by offering fixed or limited modeling components and associated functionalities. Our system sits on the other extreme, as it strives to preserve simplicity and intuitiveness, at the expense of omitting many functionalities which are presented in professional or commercial modeling systems. As our goal is toward a rapid architecture prototyping, we believe such a design choice may also be reasonable.

5. CONCLUSIONS AND FUTURE WORK

We have proposed a modeling by 3D collage framework for rapid architecture prototyping. The modeling building blocks are segmented components from a collection of existing models. Modeling operations such as affine transformations, deformation, as well

as automatic alignment are provided to make the collage process extremely easy and fun. Though simple, this powerful framework presents an intuitive and expressive user interface through which a wide variety of architectures, ranging from traditional architecture to modern ones can be constructed within a very short period of time.

There are still several limitations in our current framework. First, the perfect alignment for two incompatible faces is not considered completely solved, where the incompatibility refers to the different number of vertices on the two faces. Second, a generalized scheme is needed to support Bezier surface deformation if the corresponding face is not four-sided. Finally, more textures and more refined management of texture mapping are also needed to further enhance the quality of our composed models.

In the future, in addition to addressing aforementioned limitations, there are also a few directions that we intend to pursue. First, our model segmentation is manually done at the pre-processing time, and an automatic or more intelligent segmentation interface is very much demanded and helpful. Second, it will be very interesting to see how our system can be combined with some of the existing *image-based* modeling tools mentioned earlier, so that a prototype modeling can be made even simpler and intuitive.

6. REFERENCES

[1] P. Besl and N. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[2] M. Cabral, S. Lefebvre, C. Dachsbacher, and G. Drettakis. Structure-Preserving Reshape for Textured Architectural Scenes. *Computer Graphics Forum*, 28(2):469–480, 2009.

[3] E. Catmull and J. Clark. Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer-Aided Design*, 10(6):350–355, 1978.

[4] N. Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

[5] P. Debevec, C. Taylor, and J. Malik. Modeling and Rendering Architecture from Photographs: a Hybrid Geometry and Image-Based Approach. In *SIGGRAPH '1996*, pages 11–20, 1996.

[6] A. A. Efros and W. T. Freeman. Image Quilting for Texture Synthesis and Transfer. In *SIGGRAPH '2001*, pages 341–346, 2001.

[7] A. A. Efros and T. Leung. Texture Synthesis by Non-parametric Sampling. In *International Conference on Computer Vision*, pages 1033–1038, 1999.

[8] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools*. World Scientific Publishing Company, 1999.

[9] J. D. Foley, A. vanDam, S. K. Feiner, J. F. Hughes, and R. L. Phillips. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, second edition, 1990.

[10] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by Example. In *SIGGRAPH '2004*, 2004.

[11] X. Huang, H. Fu, O. K. Au, and C. Tai. Optimal Boundaries for Poisson Mesh Merging. In *Symposium on Solid and Physical Modeling*, pages 35–40, 2007.

[12] X. Jin, J. Lin, C. C. L. Wang, J. Feng, and H. Sun. Mesh Fusion using Functional Blending on Topologically Incompatible Sections. *The Visual Computer*, 22(4):266–275, 2006.

[13] D. Knuth. Semantics of Context-Free Languages. *Mathematical Systems Theory 2*, 2(2):127–145, 1968.

[14] J. Lin, X. Jin, C. C. L. Wang, and K. Hui. Mesh Composition on Models with Arbitrary Boundary Topology. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):653–665, 2008.

[15] M. Lipp, P. Wonka, and M. Wimmer. Interactive visual editing of grammars for procedural architecture. *ACM Transactions on Graphics*, 27(3), 2008.

[16] P. Merrell and D. Manocha. Continuous Model Synthesis. *ACM Transactions on Graphics*, 27(5), 2008.

[17] P. Muller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool. Procedural Modeling of Buildings. *ACM Transactions on Graphics*, 25(3):614–623, 2006.

[18] P. Muller, G. Zeng, P. Wonka, and L. V. Gool. Image-Based Procedural Modeling of Facades. *ACM Transactions on Graphics*, 26(3), 2007.

[19] Y. I. H. Parish and P. Muller. Procedural Modeling of Cities. In *SIGGRAPH '2001*, pages 301–308, 2001.

[20] M. Paul. Example-based Model Synthesis. In *I3D '2007*, 2007.

[21] L. Piegl and W. Tiller. *The NURB Book*. Springer-Verlag, second edition, 1995-1997.

[22] T. W. Sederberg and S. R. Parry. Free-Form Deformation of Solid Geometric Models. In *SIGGRAPH '1986*, pages 151–160, 1986.

[23] A. Sharf, M. Blumenkrants, A. Shamir, and D. Cohen-Or. SnapPaste: an Interactive Technique for Easy Mesh Composition. *The Visual Computer*, 22(9):835–844, 2006.

[24] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, 1996.

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

[26] L. Wei and M. Levoy. Fast Texture Synthesis Using Tree-Structured Vector Quantization. In *SIGGRAPH '2000*, pages 479–488, 2000.

[27] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant Architecture. *ACM Transactions on Graphics*, 22(3):669–677, 2003.

[28] J. Xiao, T. Fang, P. Tan, P. Zhao, E. Ofek, and L. Quan. Image-Based Facade Modeling. *ACM Transactions on Graphics*, 27(5), 2008.