Chapter 6

View-Dependent Texture Mapping

6.1 Motivation

Once a model of an architectural scene is recovered, the goal is to produce photorealistic renderings. A traditional approach that is consistent with current rendering hardware [1] and model file formats (e.g. VRML) involves specifying a texture map for each polygon in the model. Often, people who model architectural scenes pursue more realistic results by using texture maps cropped out of pictures of the scene itself.

Unfortunately, the texture-mapped models obtained using such piecemeal texturing methods are often visually unconvincing. One reason for this is that they fail to capture the global illumination properties of the scene: radiosity, shadows, and varying amounts of illumination from one part of the scene to another.

Another problem results when texture mapping is used to simulate more than just changes in albedo (diffuse reflectance) over a surface. This can cause particularly dull effects when texture mapping is used to simulate inhomogeneous material properties, such as would be found on a mosaic with both marble and gilded tiles, or a building façade of glass and stone. Texture mapping is also often used to give the impression of geometric detail that does not actually exist in the model, such as when an image of a rough surface is pasted onto a perfectly flat polygon. In architectural models, this practice is sometimes taken to an extreme when an image of an entire building façade, complete with window ledges, door frames, friezes, columns, or even gargoyles is used to texture map a single rectangle.

When texture maps are used to give the impression of non-diffuse material properties or geometric detail, the effect is usually convincing only for a particular point of view. As the user navigates around the scene, he or she would expect to see differing reflectance patterns from the varying material properties and motion parallax within the texture from the geometric detail. But since the same pixel values are mapped onto the scene regardless of the point of view, neither of these effects is evidenced, and the human visual system quickly notices. For architectural scenes, the experience becomes more evocative of viewing a diorama than visiting a beautiful work of architecture. This lack of realism can be referred to as the *painted shoebox effect.*

This chapter presents view-dependent texture mapping, an image-based rendering method that aims to reduce the painted shoebox effect. In this method there is not one single texture map per polygon; instead, a view-dependent combination of images, originally captured from different angles, is used to provide the reflectance for each surface in the scene.

6.2 Overview

This chapter presents view-dependent texture-mapping, an effective method of rendering the scene that involves projecting and compositing the original photographs onto the model. This form of texture-mapping is most effective when the model conforms reasonably closely to the actual structure of the scene, and when the original photographs show the scene in similar lighting conditions. In Chapter 7 we will show how view-dependent texture-mapping can be used in conjunction with model-based stereo to produce realistic renderings when the recovered model only approximately models the structure of the scene.

Since the camera positions of the original photographs are recovered during the modeling phase, projecting the images onto the model is straightforward. In this chapter we first describe how to project a single image onto the model, and then how to merge several image projections to render the entire model. Unlike traditional texture-mapping, this method projects different images onto the model depending on the user's viewpoint. As a result, view-dependent texture mapping can give a better illusion of additional geometric detail and realistic material properties in the model.

6.3 Projecting a single image onto the model

The process of texture-mapping a single image onto the model can be thought of as replacing each camera with a slide projector that projects the original image onto the model [1]. In computer graphics, this operation is known as *projective texture mapping*, which is supported in some texture-mapping hardware [39]. In the work presented in this thesis the projective texture mapping was accomplished in software.

[1]In the 1984 art exhibit *Displacements* [30], artist Michael Naimark performed such a replacement literally. He used a rotating movie camera to photograph the interior of a living room, painted the entire living room white, and then reprojected the original film onto the white surfaces with a rotating movie projector.

6.3.1 Computing shadow regions

In the general case of non-convex architecture, it is possible that some parts of the model will shadow others with respect to the camera. While such shadowed regions could be determined using an object-space visible surface algorithm, or an image-space ray casting algorithm, we use an image-space shadow map algorithm based on [56] since it is efficiently implemented using z-buffer hardware.

Fig. 6.4, upper left, shows the results of mapping a single image onto the high school building model. The recovered camera position for the projected image is indicated in the lower left corner of the image. Because of self-shadowing, not every point on the model within the camera's viewing frustum is mapped. The original image has been resampled using bilinear interpolation; schemes less prone to aliasing are surveyed in [19].

6.4 View-dependent composition of multiple images

In general, each photograph will view only a piece of the model. Thus, it is usually necessary to use multiple images in order to render the entire model from a novel point of view. The top images of Fig. 6.4 show two different images mapped onto the model and rendered from a novel viewpoint. Some pixels are colored in just one of the renderings, while some are colored in both. These two renderings can be merged into a composite rendering by considering the corresponding pixels in the rendered views. If a pixel is mapped in only one rendering, its value from that rendering is used in the composite. If it is mapped in more than one rendering, the renderer has to decide which image (or combination of images) to use.

It would be convenient, of course, if the projected images would agree perfectly where they overlap. However, the images will not necessarily agree if there is unmodeled geometric detail in the building, or if the surfaces of the building exhibit non-Lambertian reflection[2]. In this case, the best image to use is clearly the one with the viewing angle closest to that of the rendered view.

6.4.1 Determining the fitness of a particular view

We can quantify the fitness of a particular viewing angle for a particular surface using the construction shown in Fig. 6.1. Consider a small protrusion from a photographed surface that has not been modeled. This protrusion could be the edge of a window ledge coming out of the wall of a building. Projecting the real view onto the model will flatten this protrusion onto the surface of the model at the point a. Note that a virtual view from a similar direction to the real view using the real view as a texture map will still see the protrusion in approximately the correct position, despite the fact that it has been plastered onto the model. However, the virtual view should see the tip of the protrusion appear at a', but using this real view as a texture map the tip of the protrusion will appear at a. The distance, in pixels, between a and a' as projected into the virtual view can be used as a measure of the fitness of the real view as a texture map for the virtual view. Clearly, a small distance represents a more fit view. Fig. 6.2 illustrates the benefits of using the most fit view to texture-map each pixel of the model.

Note that the imaged distance between a and a' is monotonically related to the angle between the real and virtual views, although mathematically these are not the same function. An im-

[2]The images may also fail to agree for reasons not associated with the architecture itself: if there are errors in image alignment, uneven exposure between the photographs (due to different camera settings or lens vignetting), or if the pictures were taken under different lighting conditions. Also, one image may view the area to be rendered at a higher resolution than another, which will cause the resampled images to differ in their spatial frequency content. For purposes of this discussion we assume that the photographs are properly aligned, evenly exposed under the same lighting, and that each image views the area to be rendered with adequate spatial resolution to produce the desired novel view. These assumptions are most easily met when high-resolution images are taken with a calibrated camera during a single session.

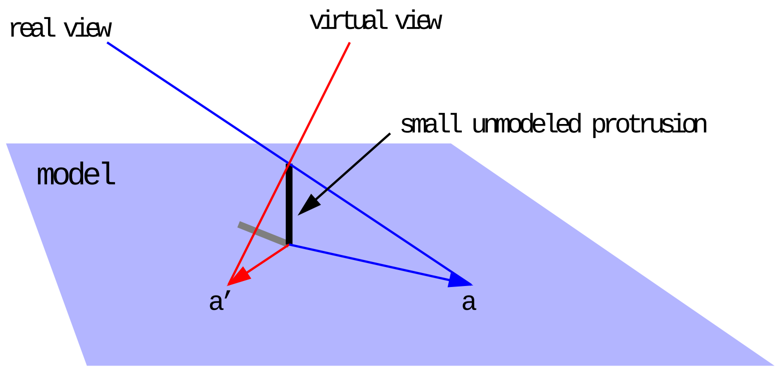

Figure 6.1: Quantifying how good a particular image is for texture mapping a piece of the model in a particular virtual view. The figure shows a very small piece of the model which is locally flat. The model, however, is only approximate, in that it does not model a small protrusion that occurs in the real scene. Since the protrusion is not modeled, using the real view as a texture map will project the protrusion onto the model at a. The virtual view, however, should see the tip of the protrusion at a'. If the virtual view were very close to the real view, a would be very close to a' and the rendering would still appear convincing. But when a and a' are far apart, as viewed from the virtual direction, the mis-projected protrusion will appear unnatural. The distance between a and a', foreshortened as seen in the virtual view, can be used as a measure of the fitness of a particular texture map. Note that this distance depends on the orientation of the surface with respect to the viewing directions.

portant difference is that the fitness function described in Fig. 6.1 greatly penalizes oblique views which cause a to fall far from the base of the protrusion, which is usually far from a'. Nonetheless, it can be useful to conceptualize the fitness function as the difference in angle between the real and virtual views.

Note also that this measure of view fitness does not consider the resolution of the original images. In particular, a very low-resolution real view from a very similar angle as the virtual view will receive a high fitness rating, even if higher-resolution views from slightly more divergent angles are available. Further work should be done to determine a sensible way to trade off image resolution for viewing angle fitness.

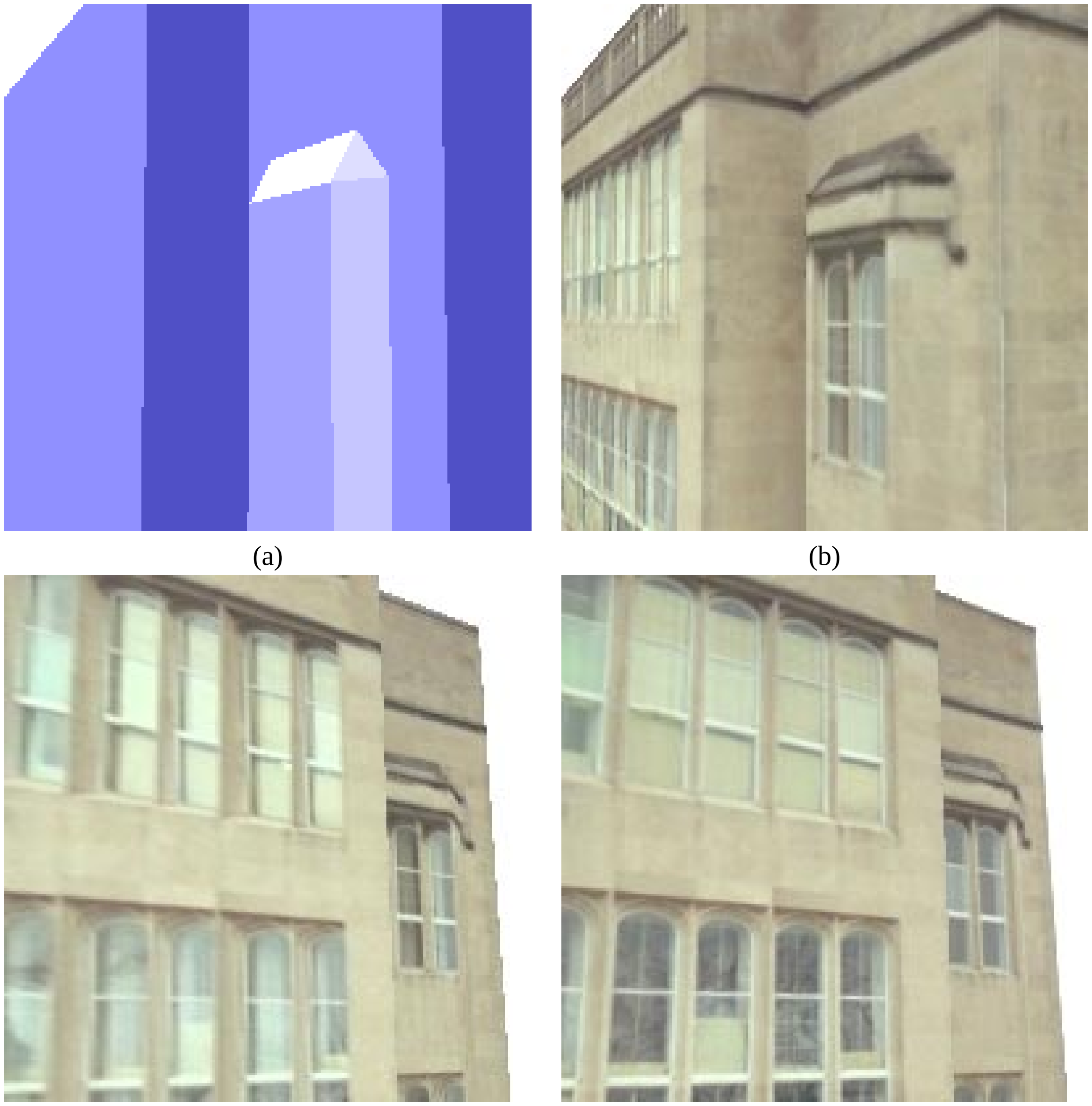

Figure 6.2: View-dependent texture mapping. **(a)** A detail view of the high school model. **(b)** A rendering of the model from the same position using view-dependent texture mapping. Note that although the model does not capture the slightly recessed windows, the windows appear properly recessed because the texture map is sampled primarily from a photograph which viewed the windows from approximately the same direction. **(c)** The same piece of the model viewed from a different angle, using the same texture map as in (b). Since the texture is not selected from an image that viewed the model from approximately the same angle, the recessed windows appear unnatural. **(d)** A more natural result obtained by using view-dependent texture mapping. Since the angle of view in (d) is different than in (b), a different composition of original images is used to texture-map the model.

6.4.2 Blending images

Using the view-dependent fitness function, it is possible to determine quantitatively the best view to use for any particular pixel in a novel rendering. Notice that the fitness function will usually indicate using pixels from different original images for different pixels in a virtual view; in fact, the fittest view can even change within the same face of the model. This effect is illustrated in Fig. 6.3.

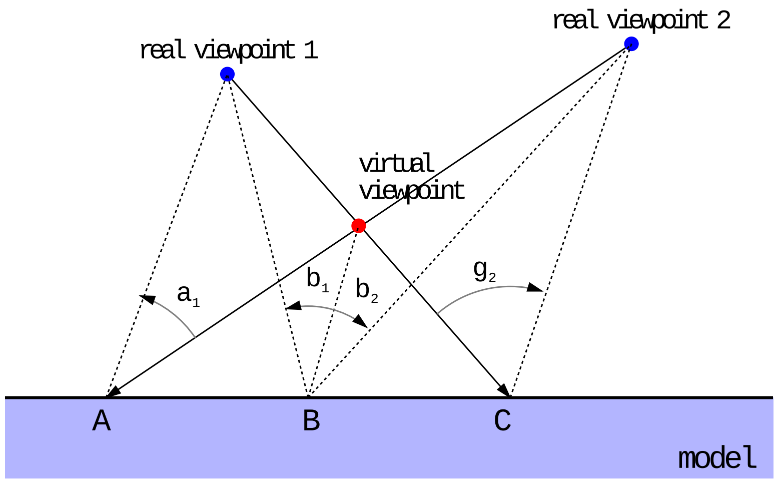

Figure 6.3: Because of perspective effects, for a given virtual view the fittest view can change across a face in the model. At pixel *A*, the best view to use is clearly view 1, since it sees the model from exactly the same angle as the desired virtual view. As a result, specular reflections and unmodeled geometric detail will appear correctly. Likewise, at pixel *C*, the best view to use is view 2. For pixel *B*, both views are equally good, according the fitness criterion presented in Fig. 6.1. One solution would be to use pixel values from view 1 to the left of *B* and pixel values from view 2 to the right of *B*. A better solution is to blend pixels from view 1 and view 2 in the area between *A* and *C* according to their relative fitness values.

If we were simply to use the most fit image at every pixel, neighboring rendered pixels may be sampled from different original images. In Fig. 6.3, this would cause pixels to the right of *B* to be texture-mapped with an entirely different image than the pixels to the left of *B*. As a result, specularity and unmodeled geometric detail could easily cause a visible seam to appear in the rendering. To avoid this problem of seams, we can smoothly blend between the two images according to their relative fitness values in the region between A and C.

6.5 Improving rendering quality

The view-dependent texture-mapping method so far presented is capable of producing several kinds of visual artifacts. This section describes several practical methods of reducing such artifacts. First, a method is described to reduce the effect of seams appearing at the boundaries of mapped images. Second, a method of removing obstructions, such as trees and cars, is described. Lastly a method of filling in regions not seen in any of the original imagery is presented.

6.5.1 Reducing seams in renderings

Even with texture blending, neighboring pixels can still be sampled from different views at the boundary of a projected image, since the contribution of an image must be zero outside its boundary. To address this, the pixel weights can be gradually reduced near the boundary of the projected images. Although this method does not guarantee smooth transitions in all cases, it tends to significantly reduce artifacts in renderings and animations arising from such seams.

6.5.2 Removal of obstructions

If one of the original photographs features an unwanted car, tourist, tree, or other object in front of the architecture of interest, the unwanted object will be projected onto the surface of the model. To prevent this from happening, the user may mask out the object by painting over the ob-

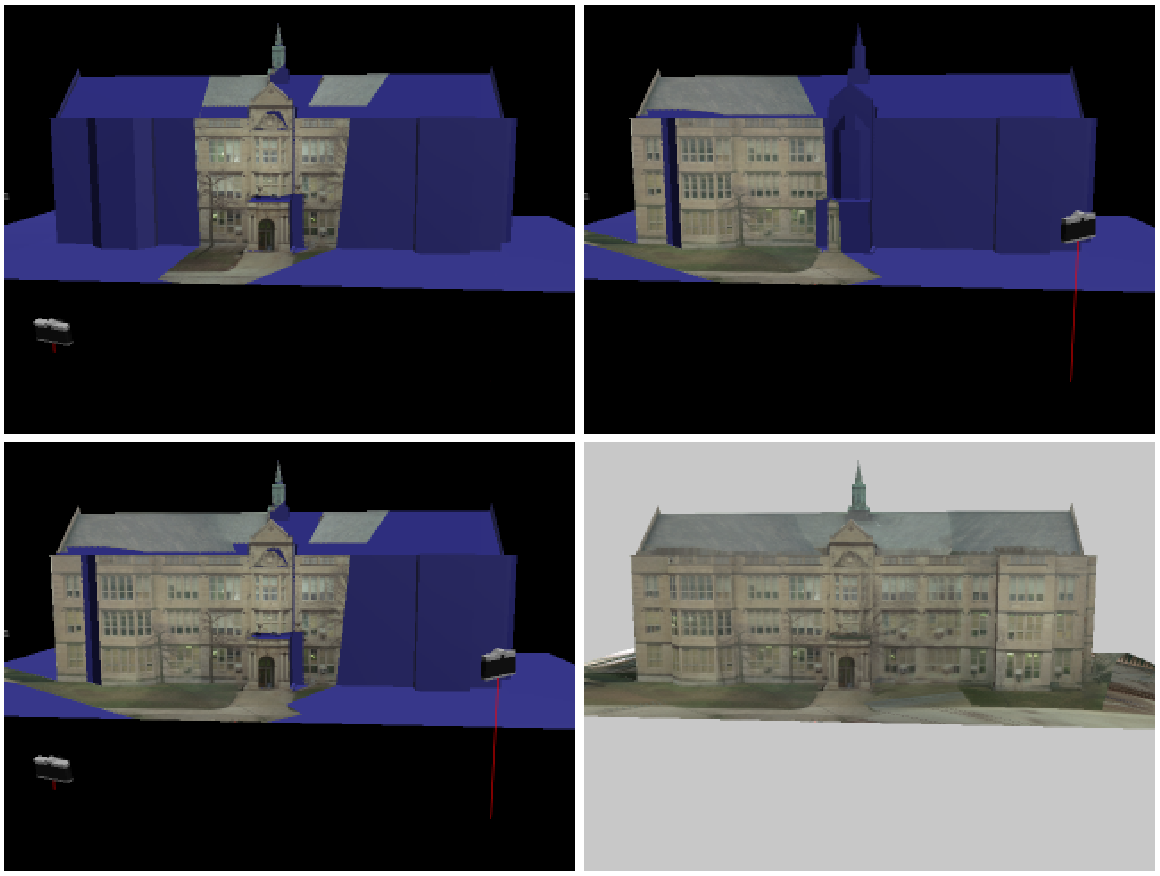

Figure 6.4: The process of assembling projected images to form a composite rendering. The top two pictures show two images projected onto the model from their respective recovered camera positions. The lower left picture shows the results of compositing these two renderings using our view-dependent weighting function. The lower right picture shows the results of compositing renderings of all twelve original images. Some pixels near the front edge of the roof not seen in any image have been filled in with the hole-filling algorithm from [57]. Some seams just visible in the roof area are due to one of the images being slightly brighter than the others; this is most probably an artifact of the "Scene Balance Adjustment" in the PhotoCD digitizing process.

struction with a reserved color[3]. The rendering algorithm will then set the weights for any pixels corresponding to the masked regions to zero, and decrease the contribution of the pixels near the boundary as before to minimize seams. As a result, regions masked out in one image are smoothly filled in by other available images when possible. Fig. 6.5 shows an image with two interposed buildings masked out of it; note that the mask need not be drawn with particular precision.

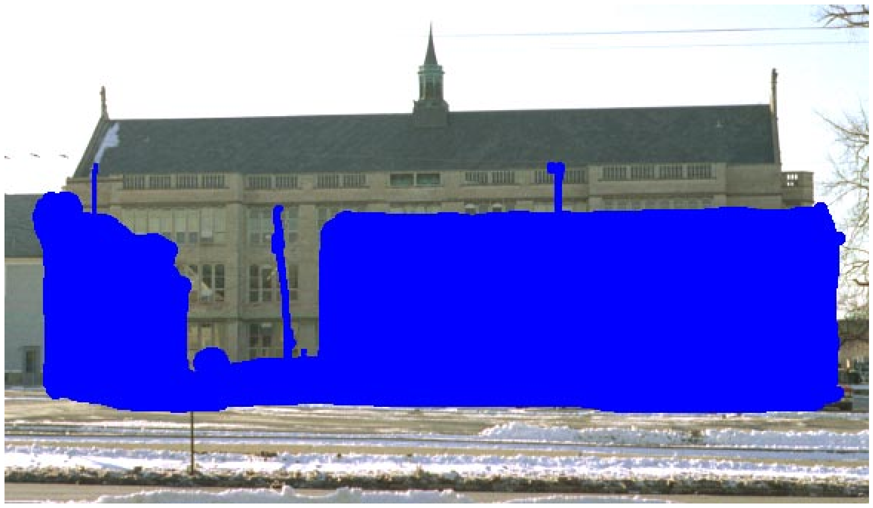
Figure 6.5: A photograph in which the user has masked out two interposed buildings that obscure the architecture of interest. These masked regions will consequently not be used in the rendering process.

6.5.3 Filling in holes

Any regions in the composite image which are occluded in every projected image are filled in using the hole-filling method similar to the one mentioned in [57]. In this iterative scheme, unfilled pixels on the periphery of the hole are successively filled in with the average values of their neighboring pixels which have color values. After each iteration, the region of unfilled pixels is eroded by a width of one pixel, and the resulting image becomes digitally spackled. Sometimes, this technique

[3]Pure blue, represented by the RGB triple (0,0,255), was used in this work since it typically does not occur in real-world scenes and is also well outside the gamut of most image sensors.

produces unattractively smeared results when a hole straddles more than one face of the model: if the two faces were shaded differently, the edge between the faces after hole-filling can look noticeably blurred.

Somewhat better results can be obtained by first only allowing the algorithm to propagate pixel values within the individual faces of the model. Thus, pixel values will not be able to cross over face boundaries. In most cases, this step fills in all the occluded pixels in the model. However, if there are some faces of the model that are not seen at all, a second pass is necessary where pixel values are allowed to creep across face boundaries in the image.

6.6 Results: the University High School fly-around

Fig. 5.14 showed a synthetic image of the high school building generated by the view-dependent texture mapping technique. This image is actually a frame from an animation of flying completely around the building. In motion, the beneficial effects of view-dependent texture mapping are more apparent. Fig. 6.6 shows every twentieth frame of the high school animation, without using the obstruction masking technique from Section 6.5.2. Fig. 6.7 shows the same animation after the obstruction masking. As a result, the surrounding trees are not pasted onto the surface of the building. See also the appendix to this thesis for information on how to access and view these animations.

6.7 Possible performance enhancements

A direct implementation of the rendering method proposed in this chapter requires performing a fitness calculation for every real view at every rendered pixel. As a result, the method is computationally intensive. This section proposes two methods of speeding up the rendering: first,

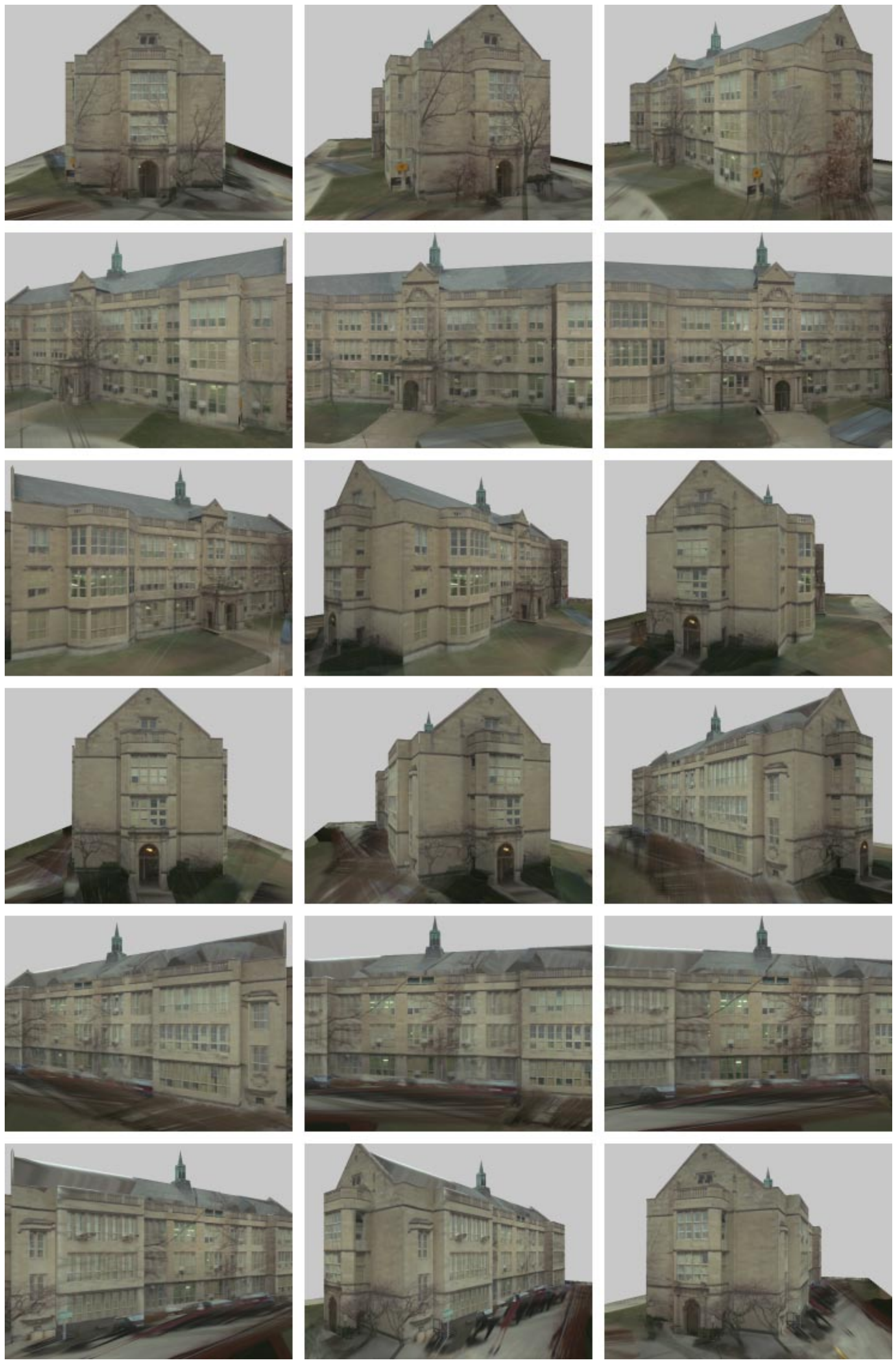

Figure 6.6: University High School fly-around, with trees. This sequence of images shows every twentieth frame of a 360-frame fly-around of the building, using twelve photographs to texture-map the model with view-dependent texture mapping.

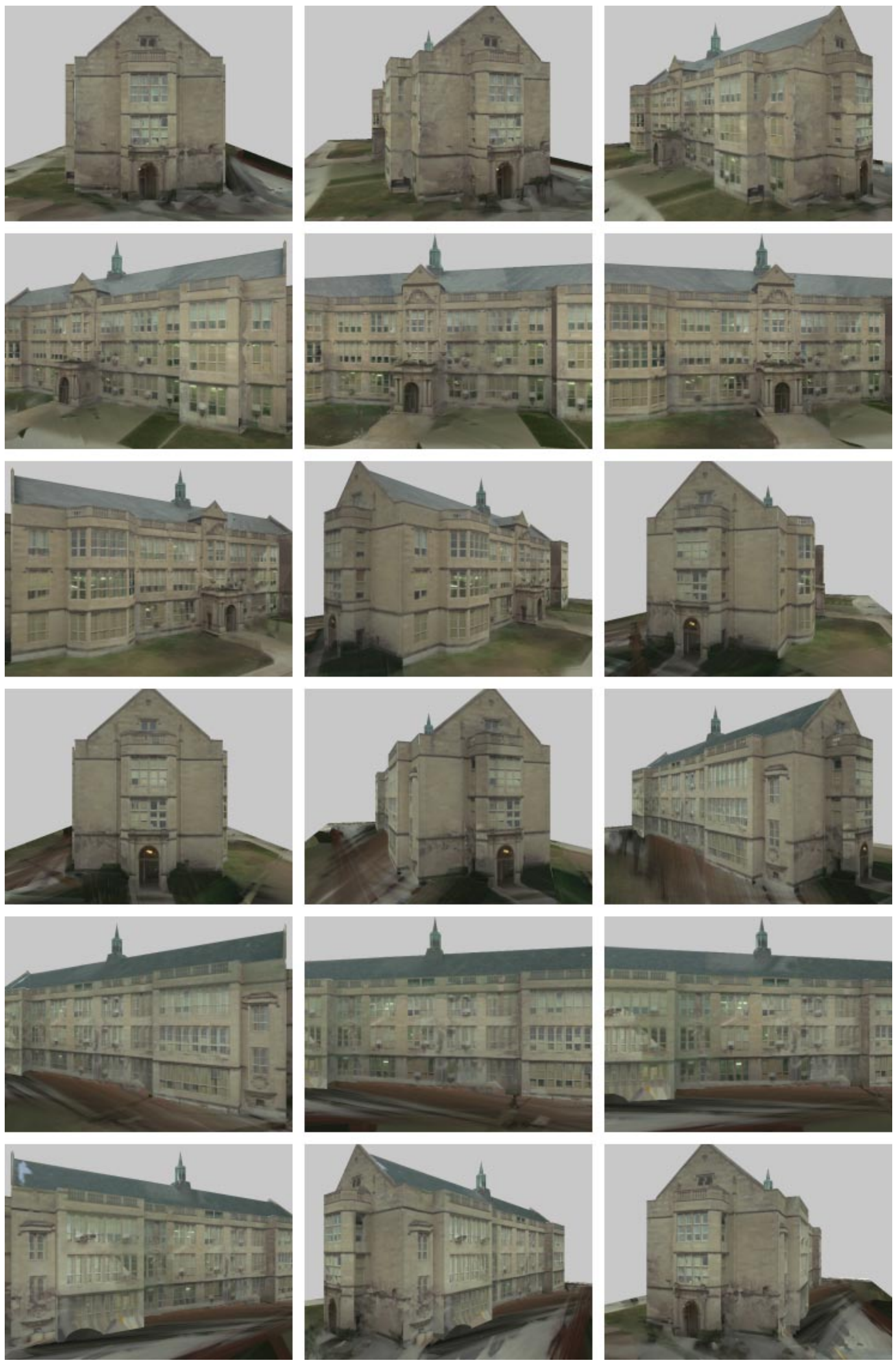

Figure 6.7: University High School fly-around, without trees. For this sequence, which is the same camera move as in Fig. 6.6, the trees were masked out of the twelve original photographs of the building using the technique of Section 6.5.2. The masked sections were then filled in automatically by the image composition algorithm.

by avoiding performing fitness calculations at every pixel, and second, by pre-selecting which real images may need to contribute to a given virtual view.

6.7.1 Approximating the fitness functions

In the discussion so far, projected image weights are computed at every pixel of every projected rendering. Since the weighting function is smooth (though not constant) across flat surfaces, it does not seem necessary to evaluate it for every pixel of every face of the model. For example, using a single weight for each face of the model, computed at the face's center, would likely produce acceptable results. By coarsely subdividing large faces, the results should be visually indistinguishable from the case where a different weight is computed at each pixel. Importantly, this technique suggests a real-time implementation of view-dependent texture mapping using a texture-mapping graphics pipeline to render the projected views, and α-channel blending to composite them.

6.7.2 Visibility preprocessing

For complex models where most images are entirely occluded for the typical view, it can be very inefficient to project every original photograph to the novel viewpoint. Some efficient techniques to determine such visibility *a priori* in architectural scenes through spatial partitioning are presented in [48].