Highlight and Reflection-Independent Multiresolution Textures from Image Sequences

Eyal Ofek Erez Shilat Ari Rappoport Michael Werman

Institute of Computer Science, The Hebrew University of Jerusalem
Jerusalem 91904, Israel. {eyalp,erezs,arir,werman}@cs.huji.ac.il

Abstract: Rendering is one of the most important tasks in computer graphics and animation. It is widely recognized that texture maps are essential for adding to the visual content of the rendered image. Extraction of textures from a single photograph poses severe difficulties and is sometimes impossible, while artificial texture synthesis does not address the full range of desired textures.

In this paper we present a method for computing high quality, multiresolution textures from an image sequence. The method has the following features: (1) it can be used with images in which the textures are present in different resolutions and different perspective distortions; (2) it can extract textures from objects with any known 3-D geometric structure; specifically, we are not restricted to planar textures; (3) removal of directional illumination artifacts such as highlights and reflections; (4) efficient storage of the resulting texture in a multiresolution data structure; and (5) no restrictions are imposed on the computed texture, which can be a constant color texture or a richly colored one.

We present an especially attractive application of our technique, in which an existing real object participates in an animation sequence and is endowed with synthetic behaviour.

Keywords: Texture, multiresolution, highlight, rendering, augmented reality.

1 Introduction

Rendering is one of the most important tasks in computer graphics and animation. In addition to the geometry of the rendered object, high-quality rendering requires the object's material properties, texture maps associated with the surfaces of the object, and algorithms to produce images given these data. It is widely accepted that texture maps are essential for adding to the visual content of the rendered image.

The simulation of light is a well-understood issue [1]. Artificial texture generation, on the other hand, still poses some unsolved difficulties. Although specific textures have been simulated successfully, artificially generated textures usually look too 'clean', even when noise is added to them using statistical methods. In practice, textures from real images (photographs, video stills, etc) or textures created using 2-D paint systems must be used in order to achieve satisfactory visual results.

There are applications which inherently require textures from real images. For example, if we want to endow a real object with a personality and orchestrate its motion with an animation software, it is very natural to require that its real textures will be available to the rendering system.

The computation of texture maps from a real image faces the following difficulties:

- The lighting conditions under which the image was taken may not match the desired illumination on the texture-mapped object. Specifically, the image usually contains specular light effects such as highlights and reflections which are not supposed to be seen when the texture is mapped onto an object.
- The geometry of the texture as captured in the image may not match the geometry of the texture as needed by the renderer's texture mapping algorithm. Textures are usually captured distorted due to perspective while they are needed in the unit square.
- The quality of the image may not be sufficient due to lack of spatial or color resolution or due to a high level of noise. The quality issue is especially true for video stills.

To counter these difficulties, one may attempt to capture the image with full control on the photographic conditions, which is often difficult to achieve. Moreover, there are very common types of textures for which controlled photography is impossible, for example when the texture is fixed in an inconvenient location (e.g. on the outside), when illumination artifacts are inherent in the object's material (e.g. the texture is an engraving on a mirror), or when a single image cannot capture the object because of other objects in its surroundings (e.g. a long wall in a narrow corridor or a building hidden by trees).

A more adequate approach is to use multiple images. This approach is also attractive because it enables us to use existing film footage which was not taken with the purpose of producing textures. However, this approach presents us with a new set of problems:

- The geometry of the texture may be different in each image.
- The resolution of the texture may be different in each image.

- Light components depending on the viewing direction, including highlights and reflections, appear in each image and in different locations.

In this paper we present a method for computing high-quality multiresolution 2-D textures from multiple images while overcoming these problems.

Related Work

Burt and Kolczynski [2] present an algorithm for fusion of several images into a single image. The algorithm is limited to images taken by a stationary camera and is mostly suited for fusion of images obtained from sensors of a different nature.

Irani and Peleg [3] create a super-resolution image from an image sequence when the relative translations between the images are known to sub-pixel accuracy. Their algorithm assumes only 2-D translations, requires a-priori determination of the final resolution and is highly computationally intensive.

Combining different resolutions in the same setting is usually done through a multiresolution representation, e.g. a pyramid or a quadtree. Berman et al [4] describe a system that uses a wavelet quadtree to economically store multiresolution information. Their application is a paint system in which new paint covers (or is blended with) former information. Their application is not suited to the case when we want to combine several images that depict the same object, since the final image obtained by their system depends on the order of fusion of the images, and since a wavelet pyramid cannot represent the intermediate data structures essential for such fusion.

For highlight recognition and removal from a single image, polarization [5] and color space techniques were proposed [6]. Polarization achieves promising results but requires photography using a special filter. The color space methods use the ideas behind Shafer's dielectric material model [7]: pixels corresponding to a dielectric material's color are concentrated near a line segment in color space, while pixels corresponding to the highlights deviate from this segment. Color space methods attempt to isolate the material color's segment. These methods do not work properly when the surface is not smooth or is highly textured, or when there is more than a single light source.

Lee and Bajcsy detected nonoverlapping highlight regions in pairs of images, by matching pixels with similar colors [8]. Their method can not distinguish between highlight and occlusion and does not specify the highlight regions very accurately.

None of the color space methods works for richly colored textures.

The Proposed Method

In this paper we present a method for computing high quality multiresolution textures from an image sequence. The input consists of a set of images and a mask on each image denoting the portion of the image from which texture should be extracted.

Obtaining such a mask completely automatically is of course a very difficult problem, whose solution is one of the primary goals of the large field of Computer Vision. Our general philosophy is to save the user as much work as possible while allowing for human intervention to correct the system mistakes. One possible scenario would be for the user to mark a set of points whose 3-D structure is known, and let the system track these points along the image sequence to produce the necessary masks. Another scenario would be for the system to automatically compute the points' 3-D structure according to well-known epipolar geometry equations. Note that it makes sense to ask users to mark an initial object, since they are usually not interested in extracting textures from *all* the objects in the images.

In the general case, tracking five points of a *known* 3-D structure along an image sequence is enough for reconstructing the geometric display transformations of the 3-D model (four points are enough for a plane). There are many automatic tracking methods and approaches, but none is absolutely reliable, especially in an environment containing highlights and reflections. A tracking method that can transform each pixel of one frame to a corresponding pixel in the next frame (e.g. optical flow) will eliminate the need of a known 3-D model, but those methods are more susceptible to illumination effects than other methods. The method described in [9] uses trifocal tensor for the tracking and is relatively stable under the basic assumptions of this paper (described bellow).

All the examples in this paper were produced by manually tracking fifteen points of a known 3-D model, on sequences not longer than 16 images. We use fifteen points in order to obtain a more reliable result than that which would be obtained using five points. A 3-D model which is not absolutely accurate (for instance ignoring round corners of a polygonal object) will not, in general, affect the tracking, but will possibly produce inaccurate local registration, only in the problematic areas.

The method possesses the following features:

- Treatment of images in which the textures are present in different resolutions and different perspective distortions.
- Extraction of textures from objects with any known 3-D geometric structure; specifically, we are not restricted to planar textures.
- Removal of directional illumination artifacts such as highlights and reflections. As far as we know, there is no other method with this capability in the presence of textures.
- Efficient storage of the resulting texture in a multiresolution data structure.
- There are no restrictions on the computed texture. It can be a constant color texture or a richly colored one.

The method can work either in a batch mode in which all the images are initially given or in an incremental mode in which the images are given one by one. Each image is warped onto the desired texture space and inserted into a hierarchical data structure according to its resolution. The color information of each image is fused and distributed at all levels of the hierarchy according to estimated quality. Directional illumination effects are removed using a robust statistics method. The latter method requires that the scene will be static; only the observer is allowed to move.

One particularly attractive application of our method is the production of animation sequences of existing objects endowed with synthetic behavior. Such application emphasizes the advantage of the multiresolution representation since each frame uses the level of detail it needs for any location in the texture.

A note on terminology: whenever we say 'texture' we mean the output of our algorithm, which actually is an image just as the input is comprised of images. We call it a texture since its purpose is to be mapped on an object in the course of rendering. The word 'image' always refers to the input to our algorithms.

2 Multiresolution Texture from Image Sequences

In this section we describe a method for computing a multiresolution texture from multiple images. There are two straightforward ways to fuse the texture from all the given images. One way is to average all the values mapped to a certain location in the texture space. Thus each texture pixel's color would be the average of its color as it is captured in the images. This leads to the corruption of the quality of the higher resolution images, e.g. when a number of far view images of a wall are given, and a single near image of a picture hanging on that wall. The second straightforward way is to choose for each texture pixel the highest resolution image which is mapped into it. This leads to conservation of noise and other changes of color resulting from camera or picture development artifacts.

Our method combines the advantages of each of the above methods. We use all the available images to determine a value in any texture point using a weighted average which gives a priority to the finer details, but does not ignore the information contained in lower resolutions. The result is more noise invariant than the second method above and without damage to the fine details as in the first one.

The method is described in two stages: first the simpler case, in which the input images are related only by 2-D affine transformations, and then the general case where they are related by 3-D projective transformations (perspective). The former case is discussed both because it is useful in its own right and as an introduction to the general case.

The justifications for the algorithm are given in the appendix. The treatment of illumination effects is described in Section 3.3.

2.1 The Quadtree Representation for Multiresolution Textures

A multiresolution texture T is a disjoint cover of the texture space (the unit square) by axis parallel constant color squares of different dimensions (pixels). In most cases the dimensions of the pixels are reciprocals of powers of two.

A natural representation for a multiresolution texture is a hierarchical data structure in which the resolution of a level is half the resolution of the one below it. In a pyramid each level constitutes a full tiling of the texture space, while a quadtree is a sparse representation in that the levels are not always full. Other variants are also possible.

The representation we use for the produced texture is a quadtree Q. The highest level (root) of Q contains a single node which corresponds to the entire texture space. The next level contains the root's four children which correspond to the four quadrants of the texture space, and so on. The lowest level nodes correspond to the smallest squares in the multiresolution texture. The quadtree is sparse since its height is lower at lower-resolution portions of the texture that it represents.

Our representation is not based on differences, e.g. wavelets or Laplacians (although we generate a Laplacian as an intermediary step). The reason is that the most common query that a texture should support is texture mapping, a query best optimized by final color values. Figure 1 shows the information contained within each node of the quadtree Q. The `value` field is the color of the node. The `certainty` field represents the amount of information gathered in the node for weighted average with a new value sampled for that node, as described in Section 2.2.1. The structure *HighlightInfo* is a temporary structure which is used only for the directional illumination removal. It is `null` after eliminating the directional light effects. The structure and its usage will be described in Section 3.3.

```
struct QNode {
    pointer to QNode          child[i]  i ∈ [1,4]
    RGB                       value
    real                      certainty
    pointer to HighlightInfo  t
}
```

Figure 1: Structure of a quadtree node.

Pointers to the node's children are stored in `N.child[`i`]`. The tree is as sparse as possible: only where higher resolution texture is required, a new level is opened. Texture regions for which only coarse information exists correspond to a shallow subtree.

2.2 Images Related by 2-D Affine Transformations

Assume that the textured object is on a plane, and that the image sequence depicting it was generated by an orthographic projection. There are cases when this situation arises in practice, e.g. when the object is relatively distant from the observer or when fusing several scanned parts of an image, each in a different resolution, into a single multiresolution image.

Denote the given set of images by $P_i, \quad i = 1, \ldots, n$. Each image is arbitrarily located with respect to the texture: it can be contained within it, contain it, or partially overlap it (see Figure 2).

Assume that the texture space is defined in some standard coordinate system (e.g. the unit square), and that the 2-D affine transformations t_i from the texture space to the image P_i are known for $i = 1, \ldots, n$ (e.g. from tracking). Computing the texture T is done in two stages:

1. Constructing the tree Q and accumulating the texture information from the images. The

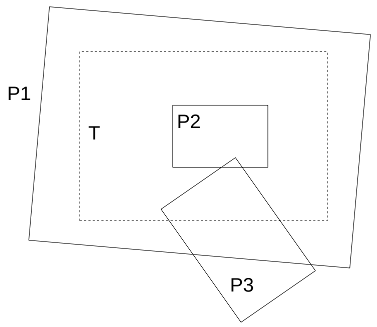


Figure 2: The geometric relations between the images and the texture space.

resolution for each point in T is determined according to the transformations t_i. The tree Q representing the texture T is expanded according to the resolution in every texture point. The texture from each image is added to Q at nodes which correspond to the observed texture resolution in the image.

2. Propagating the information up and down the tree, generating the final multiresolution hierarchical texture. This stage is necessary to obtain a tree in which each node's value is influenced by all the information stored in the tree (Section 2.2.2).

The method is described above as a batch process in which all the images are given initially. In order to express it as an incremental process in which the images are given one by one, the information propagation step should be performed after each image.

2.2.1 Constructing the Quadtree Q

This subsection presents the construction of the texture's quadtree Q along with the accumulation of the information from images. We desire a texture that contains all the information present in the images. Hence, the final resolution of any point of the texture T is the maximal resolution that the point appears in the images $P_i \quad i = 1, \ldots, n$.

The algorithm starts with a tree Q containing only the root. Using the transformation $t_i : T \rightarrow P_i$, the current resolution of T (the current level in the tree Q) is compared to P_i's resolution. If the texture's resolution is coarser, the tree is further developed until its resolution meets the image's resolution. The texture information from P_i is then added to the proper level in Q using the transformation t_i.

The texture's resolution must not be coarser than the resolution of images mapped to it, and should not be much finer (to save storage space). Achieving this goal is done by the following steps:

- Projecting a texture 'pixel' (a leaf in the tree Q) into the image space P_i using the transformation t_i.
- Examining the resulting quadrangle in the image space to decide whether a local refinement for Q is needed. The decision can be done using any combination of the following criteria:

(1) the area of the quadrangle (i.e. if it is more than 1, declare that P_i's resolution is finer); (2) the diameter of the projected quadrant (its longest diagonal); and (3) reaching a maximal resolution determined a-priori.

- If the above examination leads to the conclusion that the image resolution is finer than the local texture resolution, refine the tree (the local texture resolution) by splitting the current leaf.

After developing the tree up to the appropriate resolution according to t_i, each node N of that resolution is updated by the value of P_i at $t_i(N)$. The new value of the node is the weighted average of its previous value and the new value from P_i according to the two certainties. The certainty measures the support of a value: the certainty of the new value is one while the certainty of $N.value$ is k where $k \leq i$ is the number of values accumulated so far.

Implementation is done by calling the *TreeConstruct* procedure in Figure 3, with Q's root as an argument. The root's certainty is initialized to be one. The procedure develops Q wherever necessary and accumulates the texture information from the images. The procedure is called for all the input images.

```
TreeConstruct(QNode $N$, Transformation $t_i$, Image $P_i$) {
    if $t_i(N)$ is out of $P_i$'s area
        return
    if ResolutionReached($t_i(N)$)
        ColorsAccumulate($N, t_i, P_i$)
        return
    if N is a leaf node
        Expand($N$)
    for $j \leftarrow 1$ to 4
        TreeConstruct(N.child[$j$], $t_i$)
}
```

Figure 3: Construction of the quadtree Q.

ResolutionReached is a Boolean function which returns True if the current node N is in P_i's resolution using the above criteria, and False otherwise. The procedure *Expand* creates four children for a given leaf with certainties equal to one. The procedure *ColorsAccumulate* shown in Figure 4 updates N's value and certainty, where $P_i[p]$ is the RGB value of the image P_i at location p. The function *Bilinear* computes a bilinear interpolation of the pixels in the image near which the texture node's center $C(N)$ falls. Note that by construction the size of the texture node in the image is smaller than a pixel, hence its four corners always fall in neighboring image pixels.

```
$ColorsAccumulate$(QNode $N$, Transformation $t_i$, Image $P_i$) {

    $N.value \leftarrow \frac{N.value * N.certainty + Bilinear(t_i(C(N)), P_i)}{N.certainty + 1}$
    $N.certainty \leftarrow N.certainty + 1$
}
```

Figure 4: Accumulation of color values in the quadtree Q.

2.2.2 Propagating the Information Up and Down the Quadtree

After the procedure *TreeConstruct* has been called for all of the images P_i, all the color information is stored in the tree, but the levels have not been combined yet (Figure 5). For example, Q's root might not contain any value since no image contributed values to its level. Our goal is to obtain a tree in which each node's value is determined according to all the information stored in the tree. It is performed by first propagating the information up from the leaves to the root and then back down to the leaves. The justification is given in the appendix.

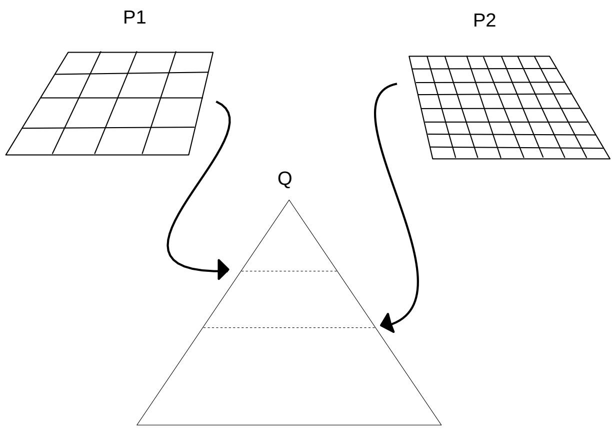


Figure 5: Two images of different resolutions inserted to Q at different levels.

The *TreeUpdateUp* procedure in Figure 6 propagates the information from the leaves up to the root. Q's root is its argument. The procedure updates each node's value to be the average of its previous value and the values of its children weighted according to their certainties. The next step is to call *TreeUpdateDown* to propagate the information from the root back to the leaves. It is called with Q's root and zero as arguments. The procedure adds to each node's value the value of its parent. Actually, *TreeUpdateUp* generates a sparse Laplacian pyramid representation of the texture while *TreeUpdateDown* transform the sparse Laplacian pyramid into a sparse Gaussian pyramid.

```
TreeUpdateUp(QNode N) {
    RGB ChildrenAverage

    if N is not a leaf node
        for $i \leftarrow 1$ to 4
            TreeUpdateUp(N.child[i])
            $N.value \leftarrow \frac{N.value*N.certainty+N.child[i].value*N.child[i].certainty}{N.certainty+N.child[i].certainty}$
            $N.certainty \leftarrow N.certainty + N.child[i].certainty$
        ChildrenAverage $\leftarrow \frac{\Sigma_{j=0}^{4} N.child[j].value*N.child[j].certainty}{\Sigma_{j=0}^{4} N.child[i].certainty}$
        for $i \leftarrow$ 1 to 4
            $N.child[i].value \leftarrow N.child[i].value-$ ChildrenAverage
}

TreeUpdateDown(QNode N, RGB ParentValue)
    $N.value \leftarrow N.value + ParentValue$
    if N is not a leaf node
        for $i \leftarrow 1$ to 4
            TreeUpdateDown(N.child[i], N.value)
}
```

Figure 6: Propagation of values up and down the quadtree.

2.3 Images Related by Perspective Transformations

This section presents the extension of the algorithms described so far to the general case, transformation under perspective projection of any given 3-D model.

Once again, the input is a set of n images P_i, $i = 1, \ldots, n$ of different resolutions, each covering an arbitrary portion of the texture T. The transformations $t_i : T \rightarrow P_i$ from the texture space to the i-th image space P_i are now mappings under *perspective* projection. The transformations t_i are easily deduced from the projection of the known 3-D model. The problem, again, is to find the resulting texture T.

The determination of the resolution is done as before, except that an image might have a changing resolution. For example, Figure 7 demonstrates how a multilevel tree is developed for a single image.

In the process of accumulating the color data from the images, the perspective must be considered. Two images might have a different certainty even if they are mapped into the same layer (images will be mapped to different layers only when one resolution at least doubles the other). Figure 8 demonstrates such a case. Both images are mapped to the same layers in the texture tree, but the left side of (a) is of higher quality than the left side of (b), and the right side of (b) is of higher quality than the right side of (a). Quality is defined in this paper to be the area in the image covered by the tree's node (other definitions are possible, e.g. Burt and Kolzynski's criteria [2]).

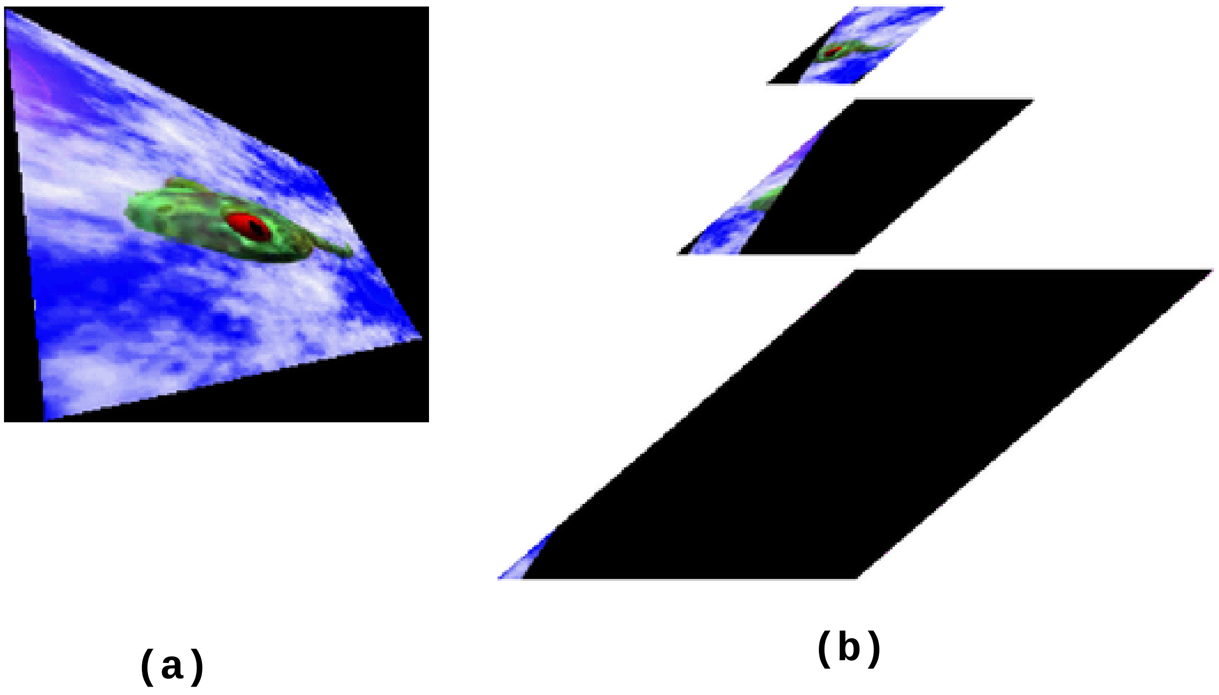


Figure 7: A texture (a) and its corresponding quadtree (b).

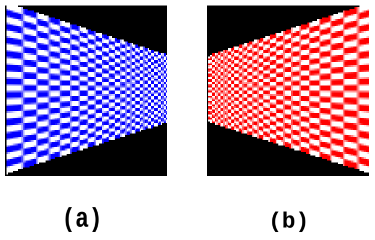


Figure 8: Projection of a uniform net from two different angles.

Consider a projection of a uniform net from the texture space into the images (Figure 8). The image's area covered by a single square of the net decreases with the quality. Figure 9 displays the texture reconstructed using only the image in Figure 8(a) and its certainty map. In the figure, higher certainty values are displayed using a brighter intensity.

The texture accumulating algorithm of the Section 2.2.1 can be adjusted to the new conditions. It is done by replacing the initial certainty of an image sample to be the area covered by projecting a tree node on P_i instead of one. See the appendix for further details. Figure 10 displays the result for the images of Figure 8. Figures 11, 12 and 13 show the same results for the fusion of two real photographs.

Non-planar surfaces result in variable resolution in a single image and are treated in the same way (Figure 14). The only difference is the computation of the transformations t_i.

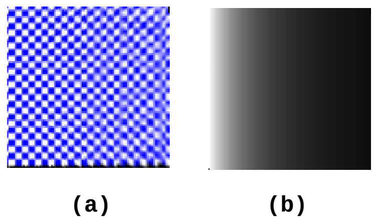


Figure 9: The texture resulting from (a) in the former figure, and its certainty map.

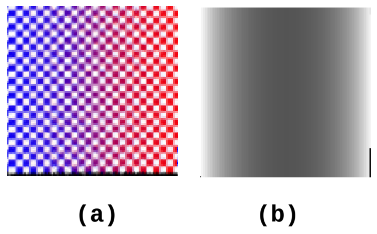


Figure 10: The texture resulting from both images, and its certainty map.

3 Removal of Directional Illumination Effects

This section presents our approach to the removal of directional illumination effects. The algorithm is capable of removing directional light from images of a surface without any assumptions on the nature of the surface texture. Specifically, it handles both very rich textures and uniformly colored surfaces.

The algorithm relies on two assumptions: that the scene is static, and that each pixel in the texture is free of directional illumination effects in most of the frames. Extraction of texture from static objects is at least as important in practice as extraction from moving or deformable objects, so this assumption does hot harm the usability of our results. The second assumption is essential, since if a pixel is covered with highlight in most of the frames it is impossible (even for a human observer) to determine its 'real' color value.

3.1 Light Reflection

Light reflection from a surface is described by the light's color and intensity and the surface's bidirectional reflectance distribution function (BRDF). The BRDF can be decomposed into three major components (Figure 15):

- Light penetrating a dielectric material and re-emerging after sub-surface scattering and refraction according to Shafer's model [7]. This component is usually called 'ideal diffuse' in the computer graphics literature and 'body reflection' in some of the computer vision literature.

  Till recently, this component was considered to be Lambertian, i.e. dependent only on the cosine of the angle between the light direction and the surface normal. Oren and Nayar [10] and Wolff [11] have shown that this component's dependence on view direction is stronger

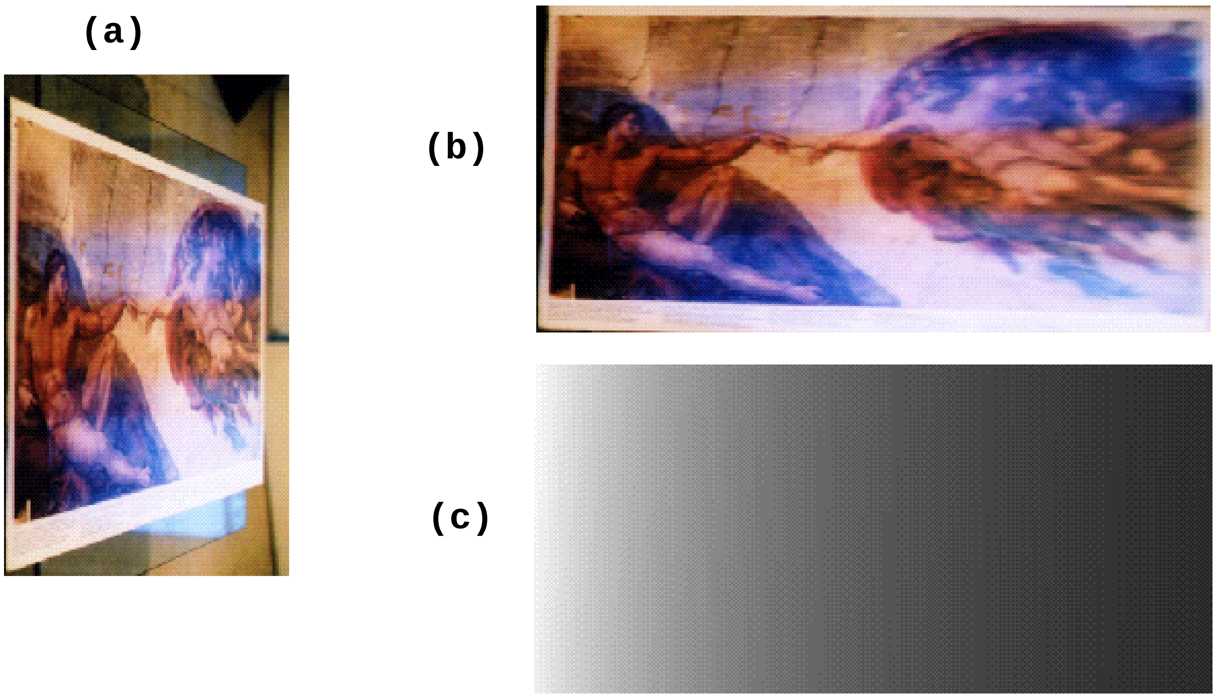


Figure 11: First original image (a), its mapping into a rectangular texture space (b), and the certainty map of the mapping (c).

the rougher the surface is. It is Lambertian only for smooth surfaces. The amount of light penetrating the surface depends on the angle of incidence and on the material's properties according to its Fresnel coefficient.

- Light which does not penetrate the material, scattering from the surface in the mirrored direction. This component is referred to as the 'ideal specular' component, and it increases in significance with the smoothness of the surface.
- The directional diffuse component. This component is present when the surface is not perfectly smooth and the material's properties enable surface reflection, depending again on the Fresnel coefficient. It can be visualized as a lobe around the mirrored direction, but is not necessarily symmetric around it, it depends on the material and the directionality of the surface's roughness.

In this paper we use the following simpler decomposition into two components:

- Directional light: light reflection which depends on the viewing direction. This component includes specular and directional diffuse reflection (surface reflection), and only a negligible portion of the ideal diffuse component.
- Indirectional light: light which does not depend on the viewing direction. This component does not always exist (e.g. in metals). It includes most of the body reflection.

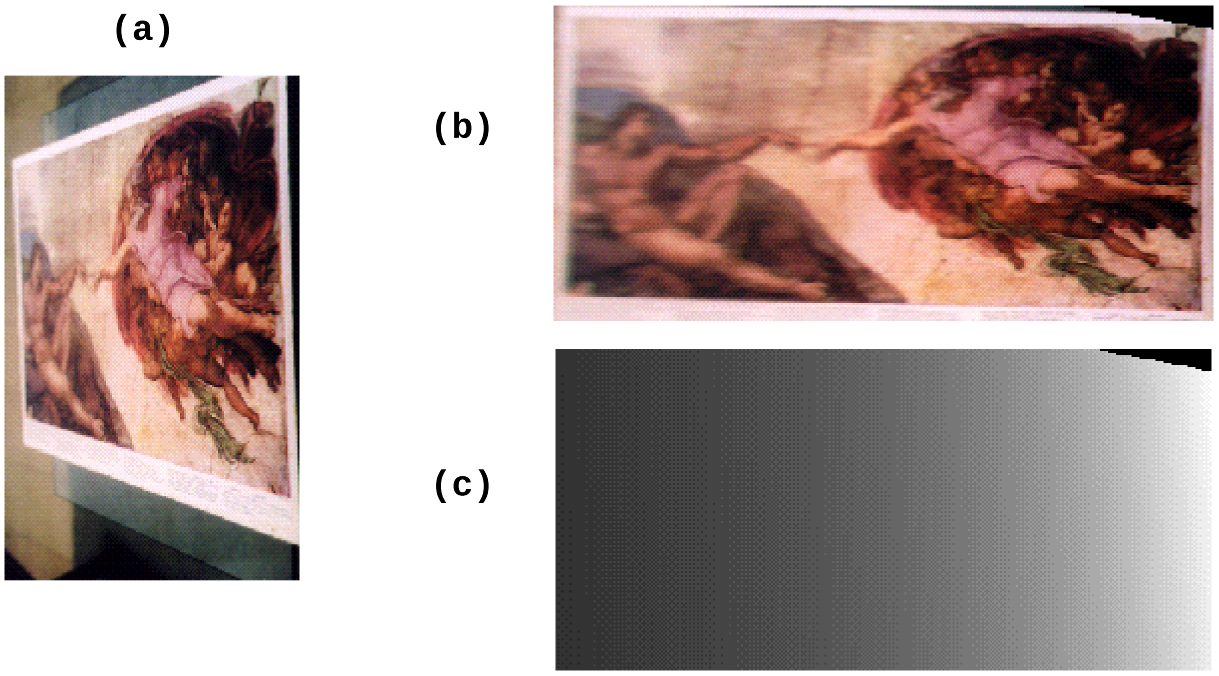


Figure 12: Second original image (a), its mapping into a rectangular texture space (b), and the certainty map of the mapping (c).

Our algorithm is capable of removing the directional illumination effects, including highlights and reflections. Note that the resulting extracted texture is not illumination-independent: it depends on the light sources intensities and colors and there are still light gradients and shadows, as in scenes rendered using the radiosity method.

3.2 Highlights and Reflections

Highlights are the reflections of a light source from an object's surface. Blake and Bülthoff [12] analyzed the behavior of a highlight resulting from a single light source for a static scene with

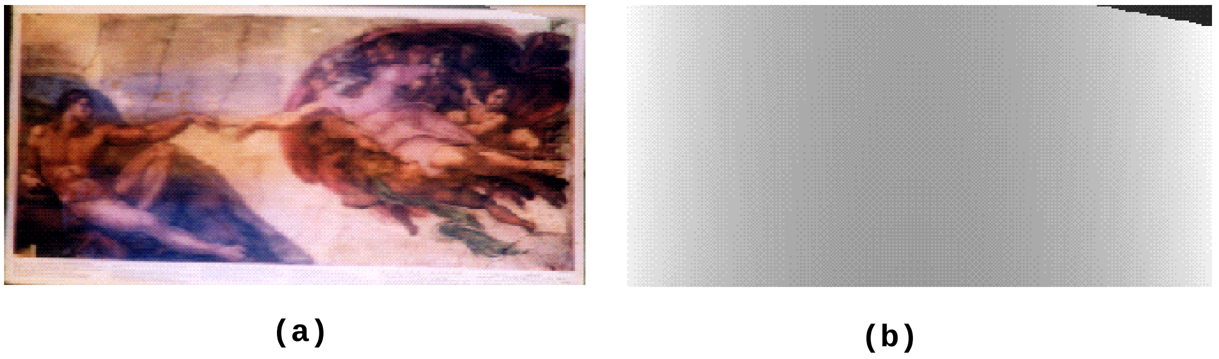
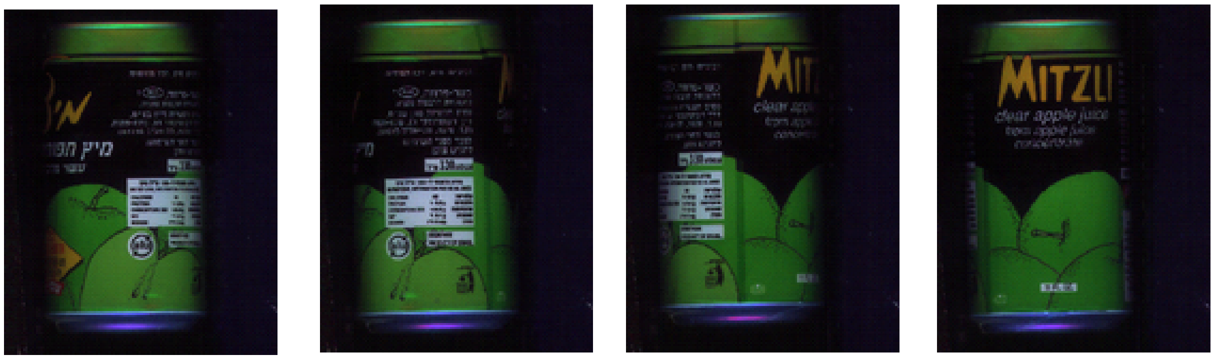


Figure 13: Resulting fusion of the two originals (a) and its certainty map (b).

(a)



(b)

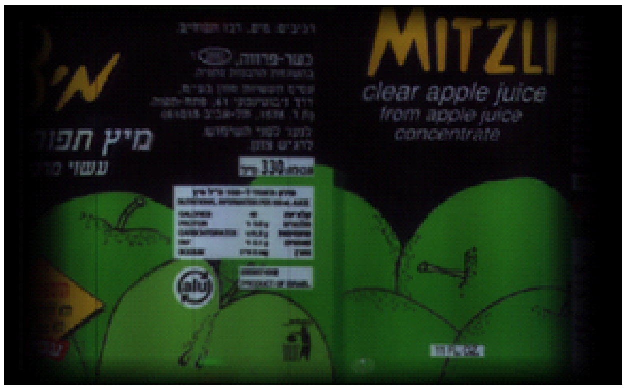


Figure 14: Four images of a soft drink can (above) and the resulting texture (below). Note that this object is not planar.

a moving observer. They expressed the highlight disparity between two images as a function of the changes in viewing direction, the local curvature and the distances of the light source and the observer from the surface.

Highlights motion in the images depends on the observer and increases in the following situations:

- When the angle between the viewing direction and the surface normal at the point increases,
- When the distance to the observer decreases,
- When the distance to the light source increases,
- When the local curvature increases.

For nearly-planar surfaces only the first three conditions apply. The above analysis holds qualitatively also for area light sources, but in this case there are additional shape changes of the highlight for observers from different viewing directions.

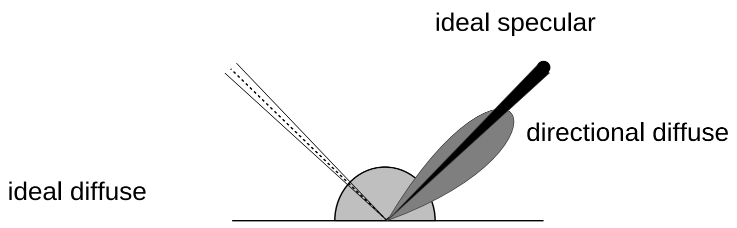


Figure 15: Components of light reflecting from a surface.

In this paper we treat reflections of other objects on the surface similarly to highlights, since in this case these objects simply serve as indirect light sources.

3.3 The Directional Light Removal Algorithm

The directional illumination removal algorithm relies on the motion differences between the scene and the directional light, assuming a static scene and a moving observer. Given sufficient viewing directions in an image sequence, the algorithm extracts the textures in the scene as if rendered using only non-directional light (for example, using a radiosity algorithm).

The meaning of 'sufficient viewing directions' is scene-dependent; it depends on all the parameters which determine the size of the highlights in the image, such as roughness, conductivity and other parameters of the materials in the scene, and the projected areas of the light sources on the surfaces.

Our basic assumption is that each part of the object of interest has been seen without significant directional light in most of the images in a given sequence. We further assume that no single angle of view is dominant in the sequence. The algorithm uses statistics to find the viewing direction independent color for each pixel of the texture covering the object of interest.

As a consequence of the above assumption and the analysis in Section 3.2, we can conclude that in 'typical' scenes under ordinary lighting conditions, only a small number of directions is required for highlight removal. The algorithm operates on the mapped images in two steps:

1. Calculation of an approximation of the view-independent color for each texture pixel.
2. Calculation of the average of $t_i(T(p))$ for every texture pixel p for which $t_i(T(p))$ is near the above value to decrease the error (recall that t_i maps the texture space into the image space).

Let us observe the color histogram that results from following one texture pixel along the image sequence. From the above basic assumption it follows that the histogram will look similar to Figure 16 (shown for a single color band, for simplicity). Most of the histogram is concentrated around the value of the view-independent color (the mode of the histogram) and the variations are due to the different amounts of directional illumination that affect the pixel in each image. There is a small peak in the bright area if the texture pixel was covered with highlight in some of the images and in a few images the texture pixel might get dark values due to insufficient sensitivity of the sensor in the highlights boundaries.

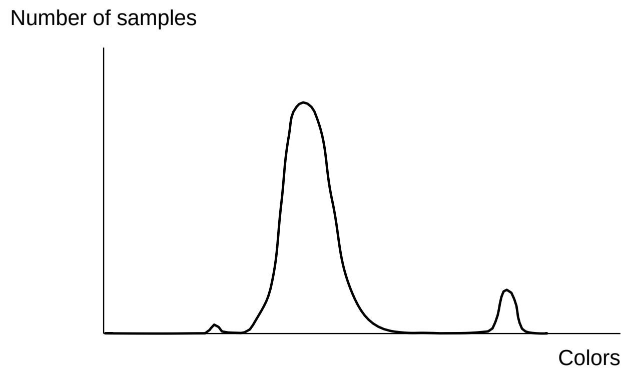


Figure 16: A typical histogram of a texture pixel along the image sequence.

Computing the histogram is a very time and space consuming task since it involves a pass on the image sequence for every texture pixel. Estimating the complete histogram, using a few randomly chosen values, is unstable since a value from an image where the texture pixel is covered by highlight might deviate the resulting estimate far from the histogram.

Our algorithm estimates the mode (the most frequent value) of the histogram using the fact that choosing a random sample of a texture pixel from the image sequence has a high probability to be in the neighborhood of the histogram's mode. It chooses m random values for each texture pixel and calculates their median. The value m is pre-defined. This median is most likely not far from the color which is responsible for the mode of the histogram. This estimate is calculated separately for each texture pixel, so the result is not smooth. Smoothing the texture corrupts its fine details and is difficult to compute in the hierarchical structure. The solution is to calculate, for each texture pixel, the average of its color values along the image sequence, for color values which do not deviate much from the median. We calculate the median and the averages separately for R, G and B. By 'color values which do not deviate much', 'much' is 10% of the R, G, and B values in our application. In our implementation, m is five. That number was chosen because it is large enough to provide a reliable approximation and small enough to store all the required information in memory. In the introduction it was mentioned that the method can work either in a batch mode or in an incremental mode in which the images are given one by one. When in an incremental mode, it first collects the first m values for each quadtree node.

Figure 18 shows a few frames before and after directional light removal. The implementation in Figure 17 is done by updating the procedure *ColorsAccumulate* of Section 2.2.1. This is the procedure which accumulates the images to their resolution level in the quadtree Q. We assume that the images P_i are given in an arbitrary order. Each node in Q is initialized with a *HighlightInfo* temporal storage place, in which m is a pre-defined value and `counter` is initialized to zero. The first m values from the images which the procedure *ColorsAccumulate* stores in Q are stored in the array $t.a$. When the array contains m values, their median is calculated and stored in $N.t.median$ of the node. For clarity we update *ColorsAccumulate* from Section 2.2.1 which ignores perspective.

4 Conclusion

We presented a method for computing high quality multiresolution textures from an image sequence. Our method treats all the problems encountered when extracting textures from image sequences, including treatment of different resolutions and perspective distortions and removal of directional illumination artifacts such as highlights and reflections. The method poses no restrictions on the computed texture; it can be a constant color texture or a richly colored one. To our knowledge, all the other highlight removal techniques which don't use special equipment (such as polarized filter) assume a constant color or a very poor texture. It can extract textures from objects with any known 3-D geometric structure, not only planar objects. The resulting texture is stored in an efficient multiresolution data structure.

The quality of the restored texture depends on the accuracy of the given 3-D model. Any deviation from the true 3-D model will result in an inaccurate mapping *for that area* and hence an inaccurate texture. It might happen that not all the directional light effects will be totally removed. It happens by violating our assumption (any texture point is not affected by directional light in most of the sequence), or if all the m first samples of a given area were under directional light effects (it is a stochastic method). The first can be handled by adding more images to the sequence taken from new angles. The second problem is simply solved by re-computing.

The extracted texture is used in order to texture map 3-D objects in still and animation image synthesis. Texture mapping is done by an algorithm very similar to Williams' mipmap algorithm [13]. The only difference from that algorithm is that filtering requires finding the neighbors of a quadtree node, a well-known quadtree operation.

A particularly attractive application of our method is the production of animation sequences of existing objects endowed with synthetic behaviour. This application is demonstrated in Figures 19 to 21. Figure 19 shows four of sixteen images of a tape-deck given as input to our algorithm. Figure 20 shows the resulting extracted textures. Note the realism of the dust on the top part of the tape-deck and the removal of the strong yellow highlights and the reflections of the table.

Figure 21 is a frame from an animation sequence in which two compact discs and the tape-deck dance to the sound of music. The texture on the table is an artificially generated marble texture.

References

[1] J. T. Kajiya. The rendering equation. *Computer Graphics*, Vol. 20 No. 2, SIGGRAPH 1986, pp. 143–149.

[2] P. J. Burt and R. J. Kolczynski. Enhanced image capture through fusion. In *Fourth Int. Conf. on Comp. Vision*, 1993, pp. 173–182.

[3] M. Irani and S. Peleg. Super resolution from image sequences. In *Int. Conf. of Pattern Recognition*, 1990, pp. 115–120.

[4] D. F. Berman, J. T. Bartell, and D. H. Salesin. Multiresolution painting and compositing. In *Computer Graphics Proceedings, Annual Conference Series*, SIGGRAPH 1994 pp. 85–90.

[5] L. B. Wolff. Scene understanding from propagation and consistency of polarization-based constraints. In *Computer Vision and Pattern Recognition*, 1994, pp. 1000–1004.

[6] G. E. Healey, S. A.Shafer, and L. B. Wolff. *Color.* Jones and Barlett Publishers, 1992.

[7] S. A. Shafer. Using color to separate reflection components. *COLOR Research and Application*, Vol. 10 No. 4, 1985, pp. 43 – 51.

[8] S. W. Lee and R. Bajcsy. Detection of specularity using color and multiple views. *Image and Vision Computing*, Vol. 10 No. 10, 1992, pp. 643–653.

[9] E. Shilat, E. Ofek, A. Rappoport and M. Werman. Trackin a rigid object along image sequences using a three-frame matching primitive. *Technical Report, Institute of computer science, The Hebrew University of Jerusalem (http://www.cs.huji.ac.il/papers/IP/index.html)*

[10] M. Oren and S. Nayar. Generalization of the Lambertian model and implication for machine vision. *Int. J. of Comp. Vision*, Vol. 14 No. , 1995, pp. 227 – 251.

[11] L. B. Wolf. Diffuse and specular reflection from dielectric surfaces. In *Image Understanding Workshop*, 1993, pp. 1025–1030.

[12] A. Blake and H. Bülthoff. Shape from specularities: computation and psychophysics. *Phil. Trans. R. Soc. London B*, 331, 1991, pp. 237–252.

[13] L. Williams. Pyramidal parametrics. *Computer Graphics*, Vol. 17 No. 3, SIGGRAPH 1983, pp. 1–11.

Appendix

The texture extraction problem is the problem of finding, for each texture pixel, the estimate color with the highest probability with respect to its colors in the images. In this section we prove that the algorithm in Section 2 finds that estimate.

Let T be the reconstructed texture. Let G be a Gaussian pyramid that represents T.

$$G^{i-1} = (G^i * g) \downarrow 2$$

where $*$ is the convolution operator, g is the Gaussian filter and $\downarrow$ is the subsampling operator. We will assume, for simplicity, that g is 1 for a node's children and 0 for any other node (so a node is the average of its four children).

As in Section 2, we start with the simple case of 2-D transformations and no perspective and generalize it later. The input consists of two images P_1, P_2 containing noise with a distribution $N(0, \sigma)$. The texture T is seen in different resolution in each image (but the texture is in a uniform resolution in each of the images). Let us now assume, without loss of generality, that P_1's resolution suits level i in G and P_2's resolution suits level $i+1$.

We can limit the discussion to the five nodes $t_1, \dots, t_4$ of level $i+1$ and their parent in the tree, t_p of level i.

Optimal Value for t_p

Two equations can be derived from the two input images:

1. ¿From P_1 we get $t_p = V_p$ where V_p is the pixel in P_1 corresponds to t_p. The measurement uncertainty is σ^2.
2. ¿From P_2 we get $t_p = \frac{1}{4}(V_1 + V_2 + V_3 + V_4)$ where $V_1, \dots, V_2$ are P_2's pixels corresponding to the texture pixels $t_1, \dots, t_4$. The measurement uncertainty is $\frac{1}{4}\sigma^2$

The measurement uncertainty is proportional to the inverse of the number of pixels (sensors) contributing to the measurement. The number of pixels represents the area in P_i corresponds to the texture covered by t_p.

We find t_p by LSE (least squares estimation), which in this case is equivalent to MLE (maximum likelihood estimation):

$$min_{t_p} \frac{1}{\sigma^2}(t_p - V_p)^2 + \frac{4}{\sigma^2}(t_p - \frac{1}{4}\Sigma_{i=1}^4 V_i)^2$$

differentiating with respect to t_p we get

$$\frac{2}{\sigma^2}(t_p - V_p) + \frac{8}{\sigma^2}(t_p - \frac{1}{4}\Sigma_{i=1}^4 V_i = 0$$

so finally we get

$$t_p = \frac{1}{5}V_p + \frac{4}{5}\frac{\Sigma_{i=1}^4 V_i}{4}$$

with the uncertainty $\frac{1}{5}\sigma^2$. Thus, each node's value is the average of its value and its children's value weighted proportional to their area's ratio.

Optimal Value for t_1, t_2, t_3 and t_4

In a similar way we obtain the following equation from P_1:

$$V_p = \frac{1}{4}\Sigma_{i=1}^4 t_i$$

with uncertainty σ^2 in measuring V_p, and from P_2 we get $t_i = V_i \quad i = 1 \dots 4$ with uncertainty σ^2.

The goal is to find

$$min_{t_j}[\frac{1}{\sigma^2}\Sigma_{i=1}^4(t_i - V_i)^2 + \frac{1}{\sigma^2}(\frac{\Sigma_{i=1}^4 t_i}{4} - V_p)^2] \quad j = 1, \dots, 4$$

Taking the derivatives according to t_i, we obtain the four new equations

$$t_j = V_j - \frac{1}{5}(\frac{\Sigma_{i=1}^4 V_i}{4} - V_p) \ \ j = 1 \ldots 4$$

Note that the quadtree Q constructed in Section 2 is a sparse Gaussian pyramid and hence updating level $i+1$ in Q is done according to level i above it.

In Section 2.2.2 we used propagation of values to obtain the final quadtree. We first propagated values from the leaves to the root to obtain a Laplacian pyramid. In a Laplacian pyramid each level stores the difference between the equivalent level and the one above in the Gaussian pyramid, so for each node

$$t_i^L = V_i - \frac{\Sigma_{i=1}^4 t_i}{4}$$

This is the result of propagating the values from the leaves to the root:

$$\begin{aligned} t_p &= \frac{1}{5}V_p + \frac{4}{5}\frac{\Sigma_{i=1}^4 V_i}{4} \\ t_i^L &= V_i - \frac{\Sigma_{i=1}^4 t_i}{4} \end{aligned}$$

The final propagation from the root down to the leaves obtains the final result:

$$t_i = t_i^L + t_p.$$

The general case of perspective projection preserves this analysis, because the certainty of each sampled measurement from an image is proportional to the area of the projected area of the relevant node on the image. The larger the area, the more image pixels (sensors) involved in the producing the node's value (Figure 22). The node's value is the average of these pixels, each with noise of distribution of $N(0, \sigma^2)$. If the area of the projection is A then the certainty is $\frac{a}{\sigma^2}$.

```
struct HighlightInfo {
    RGB             a[$i$]    $i \in [1, m]$
    RGB             median
    Integer         counter
}

ColorsAccumulate(LQNode $N$, Transformation $t_i$, Image $P_i$, Integer $m$) {
    if $N.t.counter$ < $m$
        $N.t.a[N.t.counter] = P_i[t_i(C(N))]$
        $N.t.counter \leftarrow N.t.counter + 1$
        return
    if $N.t.counter = m$
        $N.t.median$ = $MedianCalculate(N.t.a[])$
        for $j \leftarrow 1$ to $m$
            if $||N.t.a[j] - N.median||$ < threshold
                $N.value \leftarrow \frac{N.value * N.certainty + N.t.a[j]}{N.certainty + 1}$
                $N.certainty \leftarrow N.certainty + 1$
        return
    if $||\ P_i[t_i(C(N))] - N.med\ ||$ < threshold
        $N.val \leftarrow \frac{N.val * N.certainty + P_i[t_i(C(N))]}{N.certainty + 1}$
        $N.certainty \leftarrow N.certainty + 1$
}
```

Figure 17: The modified color accumulation procedure including highlight removal.

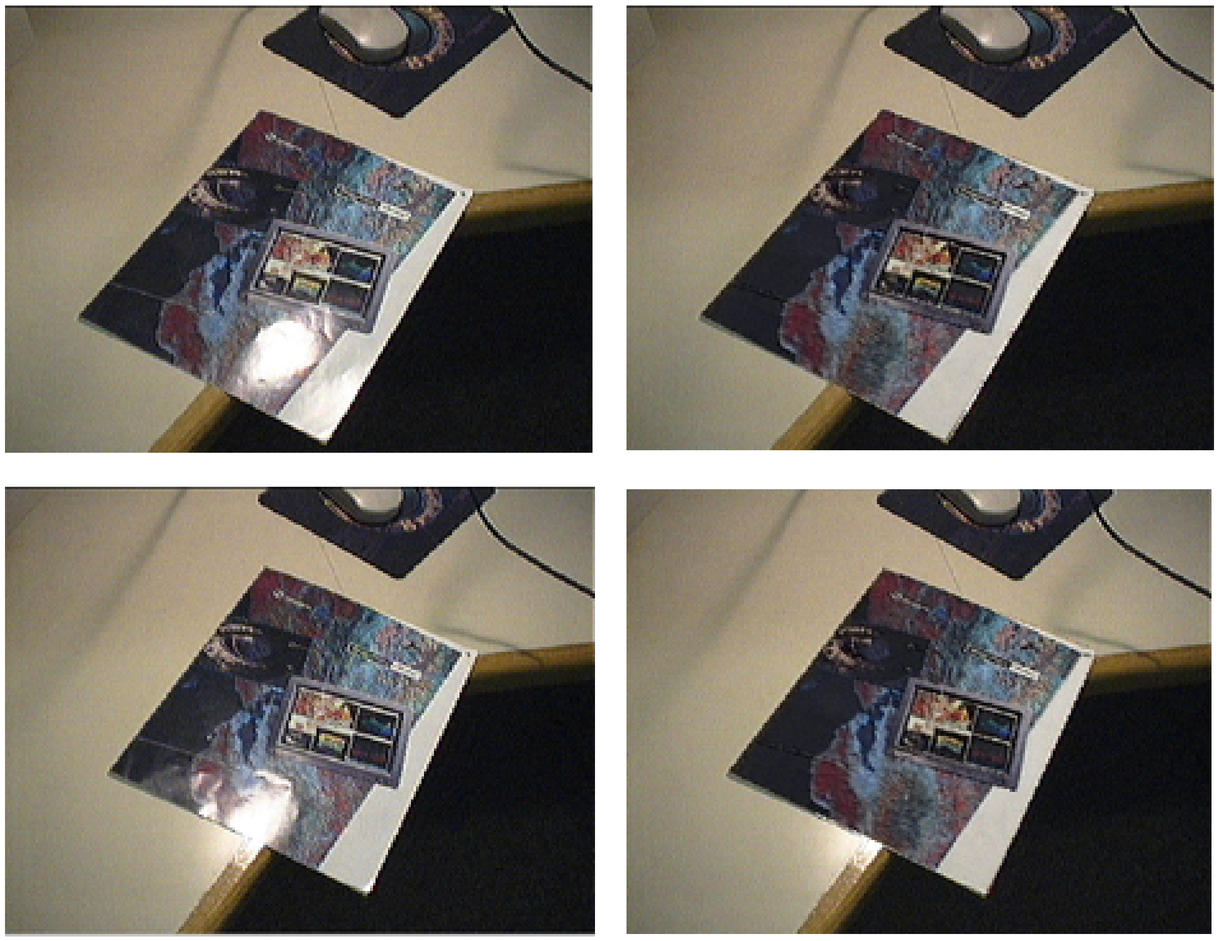


Figure 18: Several frames before (left) and after (right) directional light removal.

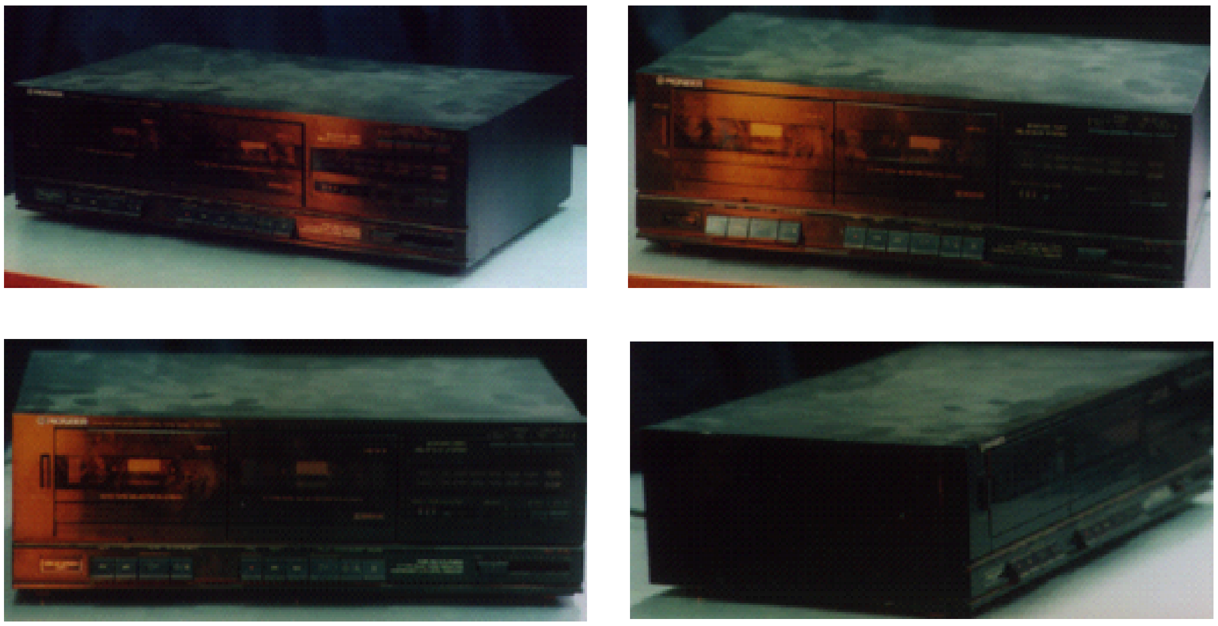


Figure 19: Four of sixteen images of a radio-tape given as input to our algorithm.

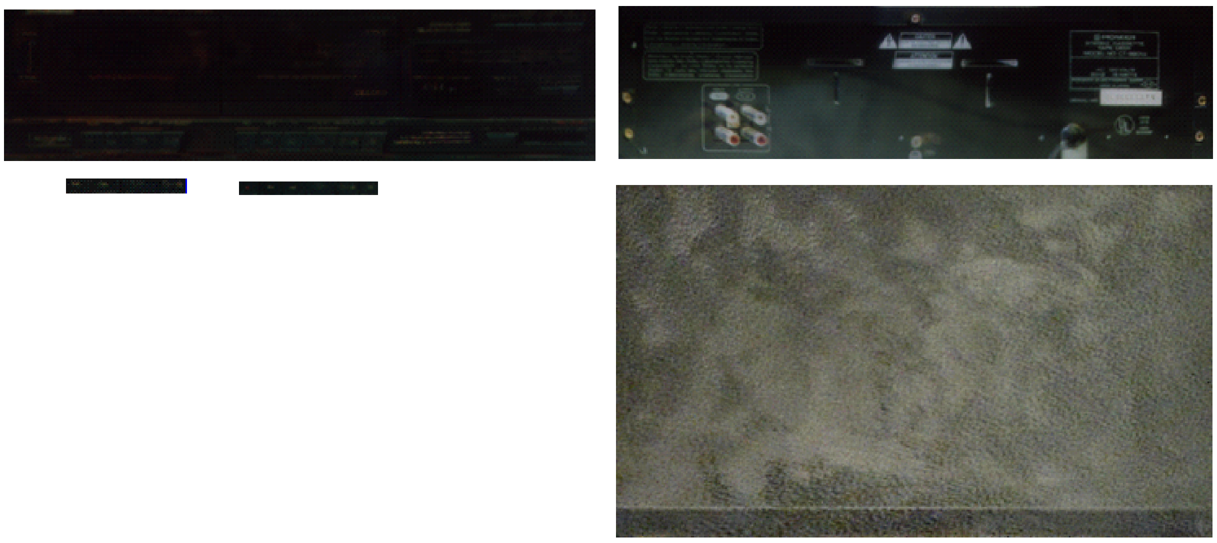


Figure 20: The textures extracted from the radio-tape.

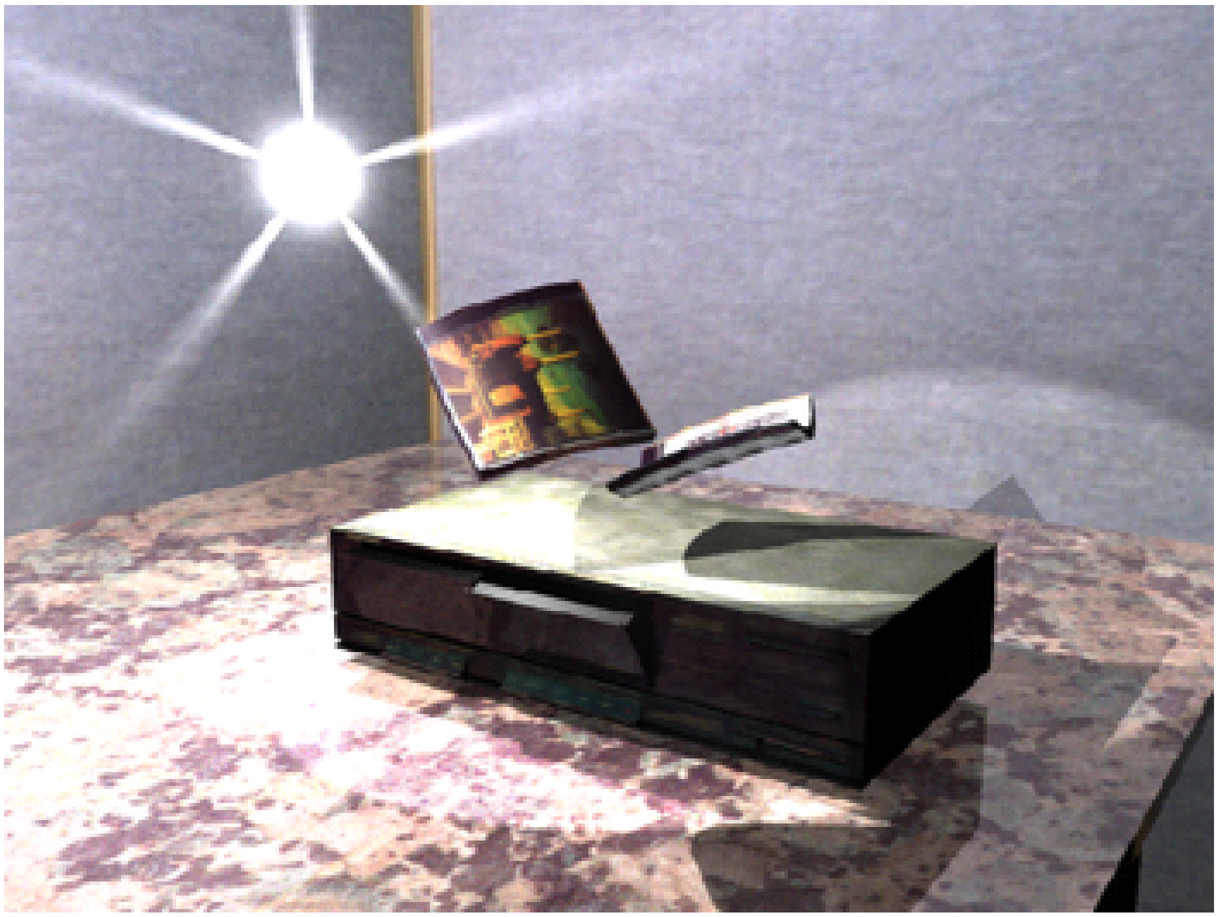


Figure 21: A frame from the ‘machine dance’ animation.

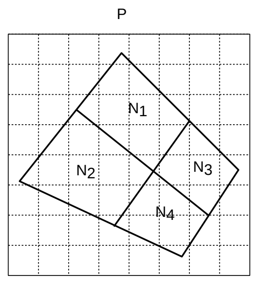


Figure 22: Texture quadtree nodes projected onto the image.