**Linear filters for halftone image processing:**

**Representations in the object and frequency space:**

Object representations have been used both in high-quality rendering as well as in real-time rendering. Both Max and Gortler et al. avoid holes in the final images by storing more than one depth layer per pixel. While these images cannot be generated directly with conventional renderers they capture the whole geometry of the object from a single point of view. When several images of the same object are available which were taken from different viewpoints, integrating these images into a single object representation requires careful registration. This process has been somewhat relaxed by Pulli et al. who resort to soft z-buffering to generate the final image. Debevec et al. tackle image registration in an interactive system. The images are projected on a crude geometric model of the object. By applying view-dependent texture mapping, high quality rendering is possible even with these crude models. The Lightfield and Lumigraph approaches sample the complete plenoptic function and use interpolation to derive new images from these samples. They are applicable both to object representation and scene representation but suffer from high memory requirements. The authors of the light field representation have also shown how to exploit texture-mapping hardware for fast rendering. Delta trees are an effective compression technique to combine multiple images of an image-based object representation into a single data structure. As more images are inserted into the data structure only those pixels are retained which show regions of the object’s surface not sufficiently sampled so far. Thereby, compression ratios of around 10: 1 are achievable in comparison to straightforward image storage.

Different types of projections have been used for storing images of objects. Environment maps (cubical, cylindrical or spherical projections) capture most of an object’s surface. With a typical perspective projection and a field of view of 90 degrees six projections are necessary to capture a cubical environment map.



Nonetheless, perspective projections are flexible enough to capture more of an object than just a single side. Consider figure 1 which shows two perspective projections of a cube. The top projection samples only one side of the cube. This is the traditional use of perspective projection. The same perspective projection can also be used to sample three (actually five in 3d) of the cube’s faces at once. In this case, the rays diverging from the eye are used to project the object instead of the rays converging to the eye. The difference between the two projections is twofold:

* on the top the viewing direction is towards the object, on the bottom it is away from the object.
* on the top clipping restricts the depth of drawn points to an interval in front of the viewpoint, on the bottom this interval is behind the viewpoint.

Using this type of perspective projection a much larger portion of the object’s surface is sampled than with a more traditional perspective projection. As a result, chances are lower to expose previously invisible portions of the object’s surface when warping only one image for a new viewpoint. When an image generated as on the top of figure 1 is warped to generate the view for point **V1,** the top of the cube will be missing. If the image generated as on the bottom of figure 1 is used instead, this surface was sampled and thus will appear in the warped image as well. When one thinks of every pixel in an image as a ray sampling the depicted object, images generated using traditional perspective projections contain no rays which appear in a perspective image of nearby viewpoints (except if moving exactly in the direction of view). In contrast, the bold ray on the bottom of figure 1 will also appear in the image for viewpoint **V1,** Figure 2 shows a perspective projection of the teapot generated in this way. Note how this view includes both the top and the bottom in a single image. As a result, fewer images are necessary in an image-based object representation to cover the surface of the object.

Layered impostors are a technique to warp a depth augmented image for a new viewpoint with hardware support on current graphics accelerators. They are based on the observation that the motion of each pixel due to changes of the point of view (the optical flow) can be approximated by rendering a number of texture-mapped polygons. The pixels in the image are grouped based on their depth and every group of pixels is drawn as a texture-mapped polygon at the corresponding distance from the new viewpoint. Depending on the number of layers used the image is warped with an accuracy increasing with the number of layers. The individual layers together approximate the volume of the viewing frustum which was each layered impostor consists of a stack of texture-mapped quadrilaterals and a texture with RGBz components per pixel. These components are stored as RGBcx textures and the correct layer content is selected from the texture with the alphafunction (a function similar to the depth test which allows to selectively draw or reject pixels based on their alpha value). Blending is disabled for drawing layered impostors so that each pixel drawn is completely opaque. When drawing the quadrilaterals the proper texels are selected by testing for a certain depth (or a) value in the texture. The quadrilaterals are conveniently specified in screenspace and transformed back into world coordinates with the inverse of the perspective projection matrix which was used to generate the texture (the inverse of the perspective projection matrix mapping the viewing frustum in world coordinates to a cube in screen coordinates).

It appears to be an open research topic how to ensure a complete object representation by a number of images of an object. Previous approaches add all available images of real world objects [7], distribute images evenly over the sphere of all possible viewing directions [3] or add all sufficiently different images [lo]. For the approach presented in this paper fully automatic representation generation was a concern and so the approach of distributing images evenly over the sphere was chosen. In order to ensure such an even distribution, the sides of a platonic solid are considered and a view is either associated with each face normal, with each direction from a vertex to the centre or the solid or with all these directions. Figure 5 shows an example image-based object representation for a teapot (all images are square to facilitate texture definition in the graphics library used). Thirty-two images have been generated in total, of which only one half is shown due to the symmetry of the teapot. On the bottom, a final image derived from this representation is shown together with the dodecahedron used to determine the viewing directions and its face normals. This image shows both how layered impostors warp an image and also produce approximate depth values at the same time. Traditional polygonal rendering (or line rendering in this example) can be mixed with images warped by layered impostors. When distributing the images evenly over the sphere final images can be generated for arbitrary viewpoints and viewing directions. In the context of flight or vehicle simulators it might be sufficient to restrict oneself to a hemisphere of viewing directions or even to images showing the object along a horizontal viewing direction. In these cases only half or even fewer images need to be stored.

**Real World Objects.** For an image-based representation of real-world objects the special perspective projections must be derived once from the available images in a preprocessing reprojection step. Several images can be considered for one projection to avoid holes in the images of the representation.

**Generating the final image.** As was demonstrated in section 3.1 and shown in figure 2 it is sufficient to warp one image out of the ones available in the representation to obtain acceptable images of the object. The image chosen is the one which was generated with a viewing direction closest to the current viewing direction. Warping only one image has a performance advantage over warping several images. In cases where small holes in the final image are less tolerable (such as the one behind the handle of the top on the bottom of figure 5), several images can be warped, so that the holes are filled in from their samples (see below).

**Avoiding excessive fillrate requirements.** Layered impostors are drawn as a series of large textured polygons. They typically cover many pixels on screen and each pixel is potentially visited as often as there are layers in the layered impostor. In order to reduce the fillrate requirements of layered impostors the images in the representation can be examined to obtain a bounding rectangle for each layer which contains the object. Only these rectangles are drawn instead of the whole slices through the frustum. As a result empty layers can be left out and the area of the other layers is drastically reduced. Figure 6 left shows an example for another object where on each layer the rectangle bounding the object is shown.

Avoiding holes. When warping more than one image it is not necessary to draw all the layers in the layered impostors of similar viewing directions. It is sufficient to draw only those layers which will actually contain the texels to close the holes in the final image. These layers can be determined in advance by rendering the layered impostor of view **V** once for each view **Vi** adjacent on the sphere of possible viewing directions (see figure 6 right) together with the layered impostor for view **Vi.** Layered impostor **Vi** is rendered with a unique identifier instead of the texture’s RGB colour. If holes appear in the warped image, layers of layered impostor **Vi** will be visible though them. These layers with identifiers appearing in the obtained image must be rendered in conjunction with the layered impostor for view **V.** Consider the sphere of possible viewing directions given on the right of figure 6. For rendering a final image the triangle containing the current viewing direction is determined (say the bold one). The layered impostor for the viewing direction closest to the current viewing direction is rendered completely **(V,).** For the two other images in the triangle **(V,** and **V)** only those layers are drawn which were found to fill in holes during preprocessing.

**Differential filters:**

**Non-linear filters:**

****

***Space-Varying Sliding-Window Filter***The general space-varying SWF (SV-SWF) inverse halftoning architecture is shown in Fig. 1. The processing steps are summarized below.

1. Reconstruction Phase: We first compute (k, l), a rough approximation of I(k, l), by applying a space-invariant SWF to the binary pixels centered at (k; l), and then compute 0(k, l) using the post-processing algorithm described in Section III. And then, pixel (k, l) is classified into one of the predesigned categories by the local variance classification scheme. Once we know which category the pixel (k, l) belongs to, the corresponding SWF weights (in Fig. 1) can be retrieved and convolve with b(k, l) and its neighbors and the reconstructed pixel \*(k; l) is obtained. We may apply another post-processing step to {\*(k, l)}; however, because {\*(k; l)} is often very close to the original gray-scale image {I(k, l)}, this additional post-processing step usually does not provide noticeable improvement.
2. Training Phase: The entire inverse halftoning procedure is made of several sliding window filters. In the training phase, we find the best weights in each SWF based on the classified training data. Every pixel in the original image is classified into one of the predesigned categories according to the local variance classification algorithm. And then, the corresponding halftone pixel b(k, l) together with its neighbor pixels inside the filter support and the original image I(k, l) constitute a block of training data. Separate SWF is designed by applying the LMS algorithm (described in Section II) to the associated set of training data blocks.

