Implementation and Applications of the Distortion Operator

Shaun Bangay
Computer Science Department
Rhodes University

Abstract

The distortion operator transforms 2D images in a manner similar to image warping or morphing, allowing source pixels to be mapped to any destination pixel. This operator can be implemented on current hardware, allowing at least one distortion per frame at interactive frame rates. Potential applications are numerous, but those described include re-mapping images for correct projection onto curved screens, correcting camera distortion from multiple sources simultaneously, and allowing constant time dynamic texturing and lighting of a static scene which is independent of geometric complexity.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Bitmap and framebuffer operations

Keywords: image warping, MMX, projection, texturing, lighting

1 Introduction

Image warping is a well known aspect of image manipulation. Warping usually implies that the components of the image retain some level of the relative relationships between the pixels, often using only affine transformations. This paper deals with the distortion operation which is intended to retain none of these inhibitions. Pixels in the image can be rearranged without constraint, duplicated or eliminated, or even replaced with pixels from other images with impunity.

This paper investigates efficiency issues with respect to the implementation of the distortion operator and considers whether it is appropriate for inclusion in the repertoire of operations associated with modern graphics languages. We investigate issues relating to the performance of hardware and software implementations of the distortion operator. Some applications of the distortion operator are described; in particular its use for projective virtual reality, its applicability for correcting distortion in video images, and its ability to re-render texture and lighting in static scenes.

2 Related Work

Texture mapping is often used for image distortion, warping and morphing, or to produce the effects of non-standard projections for applications such as environment mapping[7]. Image warping is also applied to correction of lens distortion[3] and aligning images taken from different viewpoints[4]. Further applications of image warping are described by Heckbert[8].

Catmull and Smith[2] deform textures using 3D affine transformations. The transformations are decomposed into separate horizontal and vertical stages to simplify scan-line implementation in hardware. They suggest caching intermediate values in a special framebuffer to avoid recalculating values. Fant[5] performs anti-aliasing while warping images by accumulating values across a scan-line. Pixel blending is also performed using two passes, vertically then horizontally, using a reverse mapping function. Wolberg and Boult[17] take the idea further for transformations that can be represented as separate maps, and introduce the idea of forward lookup tables for caching the functions in each of the two directions. A comparison of forward and backward mapping functions during warping with separable algorithms is given in [18].

The WarpEngine[12] is an architecture that uses images as the graphical primitives, and image warping as the operator for manipulating and combining them. The Talisman architecture[15] also uses image layers which are composited together to produce the final frame. Images may be deformed using affine transformations during this process. These transformations can reduce the need to re-render every image for every frame; often minor changes in pose can be catered for by warping the image. Seitz and Dyer[14] similarly use image morphing to produce shape transformations that look 3D using only 2D images. McMillan and Bishop[11] show the derivation of a suitable transformation to use warping to achieve image based rendering.

Heckbert[8] provides a thorough study of the image warping that can be achieved for texture mapping. Methods of filtering to reduce aliasing are also covered. Resampling of images during correction of camera distortion is described by Chiang and Boult[4].

3 Implementation

3.1 Reference Implementation

A rather inefficient and clumsy implementation of a simplified distortion operation is shown below. This form does, however, allow some aspects of the process to be clearly distinguished.

```
Image distort (Image org)
  Image chng;
  for (y = 0 .. chng.height)
    for (x = 0 .. chng.width)
      (ox, oy) = calcDistort (x, y)
      chng.setRed (x, y, org.Red ((ox,oy))
      chng.setGreen (x, y, org.Green (ox,oy))
      chng.setBlue (x, y, org.Blue (ox,oy))
  return chng
```

The distortion operation is applied for every pixel in the target image, to prevent pixels being missed due to rounding artifacts. The mapping occurs through a reverse mapping function which returns the coordinates of the point in the original image that supplies the colour to the current point in the target. In this example, the same distortion is applied to the three colour channels in the image. Separate distortions for each channel, or additional channels are possible but are not considered any further in this paper.

It is assumed for convenience that the original and target images are matched in size and colour depth.

The distortion operation results in a new image; the original image may be retained, or discarded in favour of the distorted version.

This reference implementation contains a number of performance overheads:

- It treats the image as a 2D object, with nested loops and 2D coordinates, when working with 1D coordinates would be quite sufficient.
- Calculation of distorted coordinates is repeated every time the distortion is reapplied, although there is no time dependence in the operator.
- The different colour channels are updated independently, although the same transformation is applied to each.

3.2 Lookup Table Implementation

Lookup tables are well known in computer graphics, be it for manipulating colour spaces with colour lookup tables, or for improving the performance of trigonometric functions through caching results. The lookup table in this case is large, containing the relationship between the two images for every pixel in the target.

An implementation of the distortion operation using a lookup table overcomes the limitations mentioned in the previous section.The distorted coordinates for each point in the target image are calculated during initialization of the table, and stored as the offset of the pixel within the source image. In addition to using the lookup table, the code also contains several other optimizations:

1. Transferring an entire pixel in one operation, rather than each channel independently.
2. Incremental calculation of offsets.
3. Flatten nested loops.
4. Incremental addressing of the destination image and lookup table.

Version	Random	Sphere Projection	Camera Correction
Reference	14967	568	1676
Lookup Table	125.7	56.9	54.9
1	76.5	42.2	40.9
2	66.5	36.6	35.5
3	62.2	31.9	30.8
4	61.2	30.3	29.8
Assembler	61.8	30.3	29.5
MMX	60.8	28.1	27.5
Optimized	63.7	26.7	25.0
16 bit MMX	48.5	18.9	18.2

Table 1: Processor clock cycles per pixel for implementations of distortion.

The accumulative benefits of each of these are shown in Table 1. This table shows the average number of clock cycles taken per pixel when applying distortion to a 512×512 image. The processor used is a 600MHz Pentium III. Some operations (such as memory access) depend more on the speed of the memory than the processor speed, and so the number of clock cycles may increase for faster processors. Compiler optimization is enabled for these measurements. The three distortions used are:

- Random: Randomly replaces pixels, which always performs badly in terms of memory cache access. Variance of up to 5% has been seen in these results depending on the random numbers used.
- Spherical projection: Projection onto a hemispherical screen (see section 5.1).
- Camera correction: Correction to a camera image to remove radial distortion and other artifacts (see section 5.2). The image experiences some shrinkage during this process, and unused pixels are mapped to the boundary pixels.

In practice, each pixel is represented as a 32 bit value, and transferred as such, even though currently only 24 bits are being used. This does detract partially from the benefit of moving the entire pixel in one operation, although the benefits of 4 byte alignment simplifies later enhancements.

A further enhancement to this routine would be to restrict the source image to a fixed position in memory, and store the actual address of the source byte in the lookup table. This saves referencing the value from the lookup table relative to the source image. The performance improvement in this case is quite small (under 5% for the example tested). This restriction is considered too limiting for that amount of benefit, and so is discarded.

3.3 MMX Implementation

Implementation of the distortion operation at assembler level is crucial to gaining an understanding of whether it would be feasible to implement it in hardware. The use of MMX instructions allow SIMD manipulation of 64 bit quantities (in this case, simultaneous operations on two of the 32 bit values in the lookup table, or the image arrays).

The use of the MMX registers to lookup two pixels at a time produces a small speed improvement. Writing eight bytes in one operation makes little difference, again indicating that memory bandwidth is a restriction. Unrolling the loop by a factor of four, and writing 32 bytes in sequence does eventually also produce a small speed improvement. Since the size of a cache entry is also 32 bytes, aligned on 32 byte boundaries, the performance of this code becomes sensitive to the alignment of the original images.

A further enhancement uses concurrent lookups for two widely separated portions of the target image. This allows the cache to be filled from more addresses in memory at once, and increase the chance of a cache hit. The duplication of instructions allows pairing in the U and V execution pipelines on the Pentium processor, and reduces AGI stalls when addresses used are calculated in the previous instruction[6]. The performance implications of the various assembler implementations can be seen in Table 1.

3.4 16-Bit MMX Implementation

Further performance improvements can be made by decreasing the colour resolution of the image. Instead of representing each pixel using 32 bits (of which only 24 bits are currently used), a 16-bit colour value is used. This is consistent with many modern graphics cards, which favour this colour depth for performing hardware accelerated graphics operations. Since memory access has been the bottleneck in the earlier implementations, some performance benefits can be expected by decreasing the range of memory locations which need to be manipulated.

A 16 bit representation also allows an MMX implementation to manipulate 4 pixels simultaneously. Elements in the lookup table are still 32-bit offsets into the source image, and so two MMX accesses into the table are required to retrieve the offsets for the four pixels. These are retrieved, and accumulated into one MMX register to be written in a single instruction. As before, two concurrent threads run through each half of the image to improve cache utilization.

The performance of the 16-bit implementation is shown in Table 1. With a 600MHz Pentium III processor, and a reasonably well behaved (in terms of cache access) distortion it is possible to distort over 120 512×512 frames per second. Even for badly behaved distortions, over 45 frames per second are possible, allowing the possibility of at least one distortion per frame of rendered graphics to be achieved at interactive rates on non-specialized hardware.

4 Enhancements

The basic distortion operation presented in section 3 allows for direct re-mapping of pixels in the source image to pixels in the target image. In this form it makes none of the assumptions about the relative positioning of pixels that are used by image warping to apply blending and filtering operations to smooth the destination image and reduce aliasing artifacts.

4.1 Anti-aliasing distortion

The distortion operation can be extended to include super-sampling for each pixel, and blending of the results. This permits a degree of anti-aliasing to occur, depending on the type and degree of the distortion.

The algorithm for an anti-aliased distortion is as follows:

```
Image distort (Image org,
               NumSamples,
               Offsets [])
  Image chng;
  for (y = 0 .. chng.height)
    for (x = 0 .. chng.width)
      AccumulatedColour = (0,0,0)
      for (i = 0 .. NumSamples)
        (ox, oy) = calcDistort
           ((x, y) + Offsets[i])
         AccColour +=
           org.Colour (ox, oy)
      AccColour /= NumSamples;
      chng.setColour (x, y,
        AccColour)
  return chng
```

The additions to the original algorithm include a set of offsets to the current target point at which values in the source image must be sampled, and blended together. At present each sample is weighted equally, although weights could be included to allow positions closer to the center of the pixel to count more heavily for example. The blending enhancement does complicate the process of compositing multiple distortions into one equivalent distortion operation.

This algorithm has great potential for use in concurrently distorting and blending images, and would have been so named, except for the tremendous legacy of anti-aliasing already associated with this type of transformation. Image manipulation transformations such as scaling, rotation and translation can make use of the anti-aliasing facilities to improve the appearance of the transformed image.

4.2 Multi-dimensional distortion

Transformations in graphics languages, such as OpenGL, typically operate on 3D points. Even the texture transformation matrix, which typically operates on 2D images, contains provision for 3D textures. The distortion operator is an image space operator whose 2D coordinates represent discrete pixel coordinates within the image. A discrete third dimension can easily be included with this operator, to represent a particular image in a layered stack of images.

The change in the algorithm to include this extension is shown below:

```
Image distort (Image org)
  {
    Image chng;
    for (y = 0 .. chng.height)
      for (x = 0 .. chng.width)
        (ox, oy, oz) =
          calcDistort (x, y)
        chng.setColour (x, y,
          org[oz].Colour ((ox,oy))
    return chng
  }
```

The multi-dimensional distortion operator can address multiple source images, allowing operations such as stenciling to be carried out. An application of multi-dimensional distortion is given in section 5.2.

The combination of the blending abilities of the anti-aliasing distortion, and the addressing abilities of the multi-dimensional distortion allow additional operations, such as combining textures for multi-texturing, or adding the effects of different light sources to extend the number and types of lights allowed in a scene. Section 5.3 describes some of these ideas in more detail.

5 Applications

5.1 Projection Systems

The original motivation for the distortion operation was to warp images so that they may be projected onto curved screens for use in projective virtual reality. This section gives an example of such a system, and derives the distortion operators used.

The distortion operator allows considerable flexibility when preparing projected images. The distortion can be set up to correct for the shape of the screen, with the cost of the transformation being (theoretically) independent of the curvature or distortion involved. It takes the same amount of time for orthographic or oblique projections onto one or more planar screens, as for projections onto a curved surface.

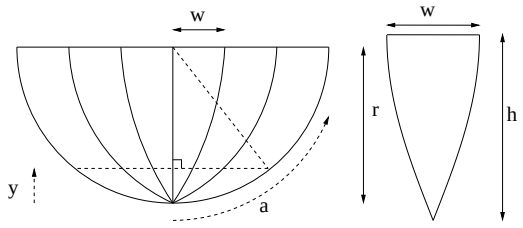
5.1.1 Projection onto a Hemispherical Screen

Screen Layout A hemispherical screen (in this case, half of a sphere) has the advantage that it is symmetrical under a range of rotations. With the viewer placed in the center of the original sphere, the hemisphere provides a field of view of 180°. By projecting onto the back of the screen from directly in front of the viewer, the entire surface of the hemisphere can be covered using a single projector.

A hemispherical screen can be approximated using a number of suitably shaped panels stitched together. Derivation of the shape of the panels is as follows:

Assume that the hemisphere is to be constructed from n panels, arranged as shown in Figure 1. The center line on each panel can be constrained to lie on the surface of the sphere. Thus the length of the line segment is a quarter of the circumference of the sphere: $h = \frac{\pi r}{2}$. The width of each panel at distance $a \in [0, \frac{\pi}{2}]$ from the tip can be determined as a fraction of the circle inscribed in the sphere at the corresponding height y, with a radius of $r \sin(\frac{a}{r})$. Thus the width $w = \frac{2\pi r}{n} \sin(\frac{a}{r})$.

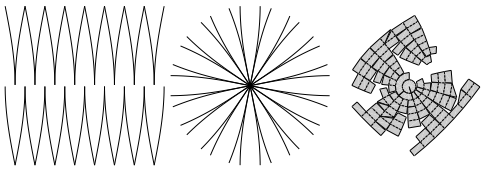
The panels can be grouped to cover the available construction material in a number of ways, some of which



(a) Sphere dimensions

(b) Panel dimensions

Figure 1: Shape of panels for a hemisphere.



(a) Parallel panel layout

(b) Centralized panel layout

(c) Panels with extra tuck

Figure 2: Panel layout strategies.

are illustrated in Figure 2. The first has all the seams running in the same direction, which makes the resulting surface prone to collapse in one direction. The second provides more symmetry in the support of the structure, but provides only one point in common to all the panels. In practice this is not a problem, as the panels are sufficiently close together in the center that the cut and seam need not go all the way to this center point.

This approximation to the surface of the sphere is appropriate for rigid materials (such as paper) which flex in one direction only. A more accurate sphere can be constructed from fabric by making tucks in the boundary arcs of each panel to reduce the length of these arcs to match that of the central line segment. This introduces substantial construction overhead, which could be more easily used to increase the number of panels, n, and improve the accuracy in that way. A layout of this kind constructed using the techniques described in [1], is shown in Figure 2c.

Screen Projection The image produced using a traditional perspective projection can be distorted into one suitable for projecting onto a hemispherical screen. Consider the scenario illustrated in Figure 3 shown looking down the y axis. The eye is placed in the center of the spherical screen, at distance d down the positive z axis. The device projecting the image onto the screen is located on the negative z axis at distance k from the origin. To satisfy the requirements of the reverse mapping used for distortion, we must find the position on the rendered image (x_p, y_p) given that we have a point

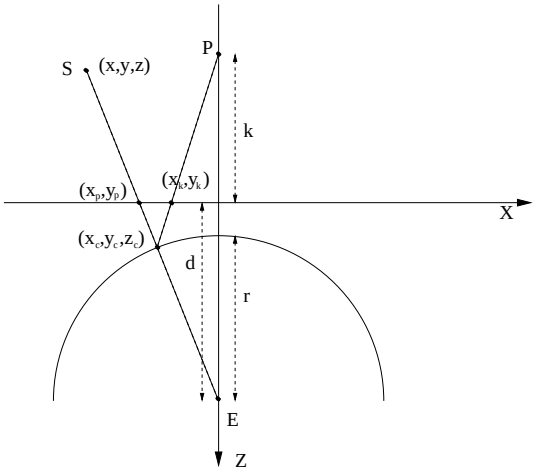


Figure 3: Producing the hemispherical distortion.

(x_k, y_k) in the image being supplied to the projector.

The ray originating from the projector and passing through the point (x_k, y_k) is:

$$R = \begin{bmatrix} x_k \\ y_k \\ k \end{bmatrix} t + \begin{bmatrix} 0 \\ 0 \\ -k \end{bmatrix}$$

for $t \geq 0$. The first point of intersection of the ray R with the screen:

$$S = x^2 + y^2 + (z - d)^2 = r^2$$

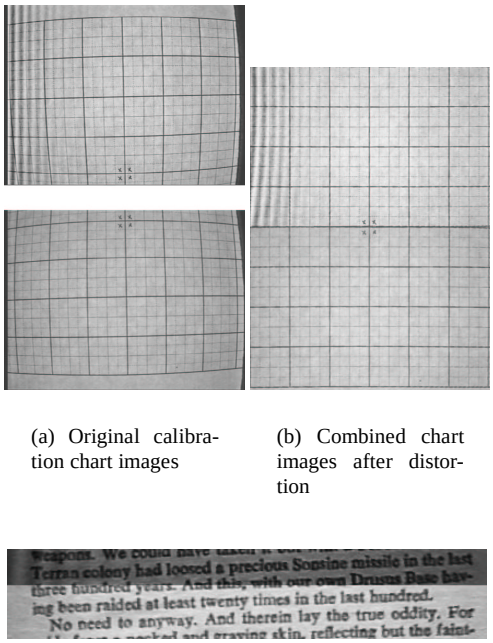
will give the coordinates of the point (x_c, y_c, z_c). This can then be projected to the point (x_p, y_p) using the relationship:

$$\begin{aligned} x_p &= d\frac{x_c}{d - z_c} \\ y_p &= d\frac{y_c}{d - z_c} \end{aligned}$$

5.2 Camera distortion

Distortion correction for images captured by camera is a popular subject for image warping. We have a more specific application that benefits from some of the extra facilities provided by an extended distortion operator. A high speed document scanner can be constructed using a housing similar to a standard flatbed scanner, but using a video camera and frame grabber for capturing the image. Such a scanner can then capture a single image in the time taken to record a single frame (less the $\frac{1}{25}$s). The disadvantage is the low resolution of the camera image (768×576 pixels), and the distortion introduced into the image by the optics of the camera. The purpose of the scanner is to produce a mechanism for capturing popular printed literature for later manipulation online.

After some experimentation, we decided that the resolution of a single camera is too low for comfortable reading. Two cameras can be used to double the effective area, without significant increase in the time to capture a page. Combating image distortion then becomes



(a) Original calibration chart images

(b) Combined chart images after distortion

(c) Sample scanned text (shown along seam)

Figure 4: Correction of camera distortion.

essential for ensuring proper registration of the two images corresponding to the two halves of a single page. This correction is also necessary if further processing (such as OCR) is to be performed.

The relationship between a point in the captured image (X_f, Y_f) and a point $(X_w, Y_w, 0)$ on the page being captured is given by (1)[3].

$$\begin{bmatrix} d_x' \frac{N_c}{N_f}(X_f - C_x)(1 + \kappa_1 r^2) \\ d_y(Y_f - C_y)(1 + \kappa_1 r^2) \end{bmatrix} = f \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \end{bmatrix} \quad (1)$$

$$\text{where } \begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X_w \\ Y_w \\ 0 \end{bmatrix} + T \quad (2)$$

Calibration of the camera is required to determine values for the various variables given in this equation. These are found by fitting equation (1) to known values of X_f, Y_f, X_w and Y_w, measured from a calibration image. The camera calibration package by Reg Willson[16] is used to choose optimal values for the parameters. A distortion map is created that corrects distortion, and combines the two camera images, both in a single step. The upper half of the corrected image is mapped to the distorted image from the first camera, the bottom half to that of the second camera. This can be achieved with a 3D distortion, with the third dimension used to select the source image.

The results of using the distortion operation to correct camera distortion are shown in Figure 4. The calibration

charts are shown before and after the distortion operation has corrected for camera distortion. A sample of text from a book[10] scanned in this way is shown (the seam can be clearly seen because the two cameras have different brightness and contrast settings). Inexpensive hardware can be used because physical alignment of the two cameras is unimportant since the distortion operation includes the rotation (R) and translation (T) operations to reposition the two images. The cameras need only be positioned to capture as much of the image as possible.

5.3 Static rendering

One of the original goals of image warping is to transform a single texture so that it matches the screen coordinates of the polygon onto which it is mapped[2]. This is possible with distortion as well, but the cost of setting up a distortion for a single polygon is excessive in light of the rendering abilities of modern hardware. However, it is possible to set up a distortion that can perform texture mapping for the entire scene in a single step. This section describes this facility, and how it is possible to include lighting, bump mapping and simple environment mapping at the same time.

The transformation from a point in screen coordinates on the surface of a polygon to the colour value in the associated texture map contains a number of implicit and explicit steps; mapping via the transformations applied to the vertices of the polygon, back to object coordinates, then through texture warping to the coordinates of the texture image. Having performed this operation once during rendering of a scene, the results can be cached in the lookup table of a distortion map, and the texture mapping repeated by applying the distortion operation.

Re-rendering the image may seem pointless, but in practice it can find a multitude of applications. The distortion operation can be reapplied with a different texture as the source, multi-dimensional distortions can allow changes in selected textures, or an additional distortion can be preapplied to the texture to produce an animated texture. Since the distortion operation works in image space, it is independent of the complexity of the scene. Highly detailed static backdrops can be created, and changes made interactively to the surface textures without re-rendering. This has applications in computer games, producing realistic backdrops with effects such as animated water surfaces, pre-rendering panoramas[13], or for enhancing the primitives when working with image based rendering[15], to name but a few.

The texture distortion is implemented by identifying the (s, t) texture coordinates of each screen point within the texture map, and associating this value with the coordinates of that point within the frame buffer. In practice the actual position of the texel within the texture map is mapped from the screen coordinate.

Texture is not the only application to which distortion can be applied in static scenes. Lighting operations involve a mapping between the position of a pixel on the screen, and the colour for that pixel, via the mechanism of the surface normal, lighting vectors and illumination equation. A distortion operation can be used to cache the surface dependent portion of this, while still allowing certain lighting parameters to be changed, and reapplied in constant time. Thus detailed backgrounds can be created, that can be matched to the same dynamic lighting situation applied to objects rendered in the foreground.

Illumination expressions are usually of the form given in (3), where N is the surface normal vector, V the vector toward the viewer, and L and R depend on the position of the light source. The various ambient, diffuse and specular factors will depend on properties of the surface and light source.

$$I(N) = I_a + (N \cdot L)I_d + (R \cdot V)^n I_s \qquad (3)$$

By assuming that the light sources and viewer are infinitely far away, the vectors V, L and R are independent of the position on the surface, and illumination is a function only of the normal to each point on the surface. By requiring that the normal vector be normalized, it contains only two degrees of freedom, and so a mapping can be created between the screen coordinates of a point, and its corresponding normal vector (through the mechanism of a distortion operation). In fact, the original normal vector is not required, only the coordinates of a point within the light map created by projecting (3) onto a 2D image.

The effects of other lighting equations can be simulated by changing the light map. Figure 5 shows some variations on non-photo-realistic cartoon rendering. Diffuse and specular effects are also shown in the same figure. The light maps used are inset into each image. The mapping from the normal vectors on the surface of a unit sphere to the light map coordinates is via a mapping which places the point $(0, 0, 1)$ at the center, with the point $(0, 0, -1)$ mapped to the boundary, and with intervening points on the radial lines with the same $\frac{x}{y}$ gradient. The mapping is distorted to make full use of the rectangular area of the light map.

The scene shown in Figure 5 uses over 250 000 polygons, which for a partially covered 512×512 image means that there are more polygons than pixels. In complex scenes with even higher polygon counts, the ability to limit graphical operations such as texturing and lighting to a per pixel basis, rather than per polygon, would be a substantial performance advantage.

For situations with local viewers or lights, it is possible to use the distortion operator to map screen coordinates to the pair $[(N \cdot L), (R \cdot V)]$ in a manner analogous to the so called 'cheap Phong' shading.

Since the distortion map is providing access to the normal vectors, it becomes possible to exploit these vectors for other applications. Bump mapping can be applied by providing a perturbation to the normal vectors, either by modifying the distortion mapping, supplying an additional distortion, or re-mapping the image accessed by the distortion operation. A variation on environment mapping can be achieved by supplying a texture containing an image of the surrounding environment instead of a light map.

Figure 6 shows images rendered using bump maps, and a pseudo-environment mapping process (the environment map does not cover the full 360°, and is not distorted correctly) to demonstrate this effect.

Access to multiple texture maps during the distortion can be achieved using a multi-dimensional distortion. Lighting and texturing can also be combined using the blending facilities of an anti-aliased distortion as shown in Figure 6.

6 Conclusion

Distortion is an extremely useful facility, and worth including in the toolbox of a graphical language. We demonstrate that it can be applied at interactive frame rates, even on non-specialized hardware. Extensions to the distortion operation exist that duplicate and extend the abilities of other well understood operators. The distortion operator can be easily implemented in hardware allowing acceleration of operations that use it.

A number of applications which use distortion are described in this paper. These are immediately applicable to modern virtual reality systems and graphics architectures, and can offer significant advantages over other rendering strategies. They are merely the most relevant applications amongst those that have been explored. Many more lie undiscovered.

References

[1] Shaun Bangay, From Virtual to Physical Reality with Paper Folding, *Computational Geometry: Theory and Applications*, 15 (1-3), 161-174, February 2000.

[2] Edwin Catmull and Alvy Ray Smith, 3-D Transformations of Images in Scan-line Order, *SIGGRAPH 80*, 14 (3), 279-285, July 1980.

[3] M. Chiang and T. Boult, A Public Domain System for Camera Calibration and Distortion Correction, Tech. Rep. CUCS-038-95, Columbia Univ., Dept. of CS, and Lehigh Univ., Dept. of EECS, available via the WWW at `ftp://ftp.cs.columbia.edu/pub/chiang/calib.ps.gz`, Dec 1995.

[4] M. Chiang and T. Boult, The Integrating Resampler and Efficient Image Warping, *Proceedings of the ARPA Image Understanding Workshop*, pp. 843-849, February 1996.

[5] Karl M. Fant, A Nonaliasing, Real-time Spatial Transform Technique, *IEEE Computer Graphics and Applications*, 6 (1), 71-80, January 1986.

[6] Agner Fog, How to Optimize for the Pentium Family of Microprocessors, available via the WWW at `http://www.agner.org/assem/#optimize`, July 2000.

[7] Paul Haeberli and Mark Segal, Texture Mapping as a Fundamental Drawing Primitive, available via the WWW at `http://www.sgi.com/grafica/texmap/index.html`, June 1993.

[8] Paul Heckbert, Fundamentals of Texture Mapping and Image Warping, Master's Thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, June 1989.

[10] Arthur H. Landis, *Home - To Avalon*, Daw Books Inc., New York, November 1982.

[11] Leonard McMillan and Gary Bishop, Shape As A Perturbation To Projective Mapping, UNC Computer Science Technical Report TR95-046, University of North Carolina, available via the WWW at `http://www.cs.unc.edu/%257Eibr/pubs/mcmillan-shape/shape.ps.gz`, April 1995.

[12] Voicu Popescu, John Eyles, Anselmo Lastra, Joshua Steinhurst, Nick England and Lars Nyland, The WarpEngine: an Architecture for the Post-Polygonal Age, *SIGGRAPH 00*, New Orleans, LA USA, Pages 433 - 442, July 2000.

[13] Matthew Regan and Ronald Pose, A Low Latency Virtual Reality Display System, Technical Report 92/166, Department of Computer Science, Monash University, Monash, 1992.

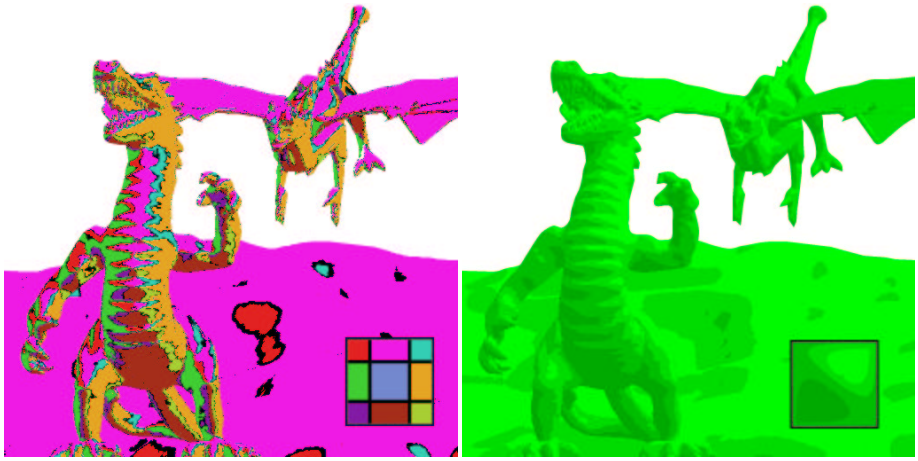
[14] Steven M. Seitz and Charles R. Dyer, View Morphing, *SIGGRAPH 96*, New Orleans, LA USA, 21-30, August 1996.

[15] Jay Torborg and James T. Kajiya, Talisman: Commodity Realtime 3D Graphics for the PC, *SIGGRAPH 96*, New Orleans, LA USA, Pages 353-363, August 1996.

[16] Reg Willson, Tsai Camera Calibration Software, available via the WWW at `http://www.cs.cmu.edu/afs/cs.cmu.edu/user/rgw/www/TsaiCode.html`, October 1995.

[17] George Wolberg and Terrance E. Boult, Image Warping with Spatial Lookup Tables, *SIGGRAPH 89*, 23(3), 369-378, July 1989.

[18] George Wolberg, H.M. Sueyllam, M.A. Ismail, K.M. Ahmed, One-Dimensional Resampling with Inverse and Forward Mapping Functions, *Journal of Graphics Tools*, 5 (3), 11-33, 2000.



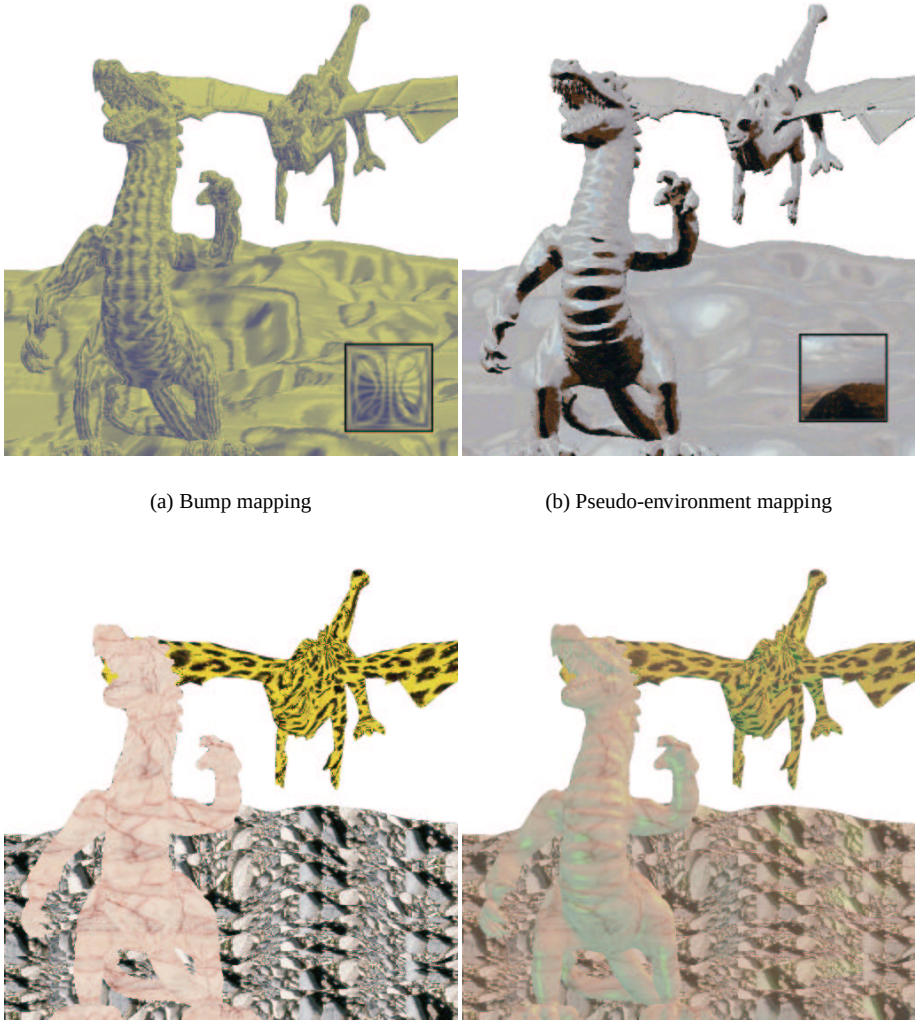
(a) Cartoon effect with simple light map

(b) Four colour cartoon shading

(c) Light map with three diffuse sources

(d) Light map with green specular term

Figure 5: Static scenery lit using distortion [Dragons from 3dcafe.com].



(a) Bump mapping

(b) Pseudo-environment mapping

(c) Multiple textures with a multi-dimensional distortion

(d) Lighting and texturing using an Anti-aliased, multi-dimensional distortion

Figure 6: Texturing and special effects using distortion [Dragons from 3dcafe.com].