School of Computer Applications

DUBLIN CITY UNIVERSITY
Ollscoil Chathair Bhaile Átha Cliath
Dublin City University, Glasnevin, Dublin 9, IRELAND.

3D Perspective Texture Mapping of Polygons in Real Time

Michael McMahon
Steven Collins

CA-1197

Working Paper Series

3D Perspective Texture Mapping of Polygons in Real Time.

A WORKING PAPER SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE (COMPUTER APPLICATIONS).

BY: MICHAEL J MC MAHON, B.E.(MECH) M.I.E. M.B.A.

SUPERVISOR: DR STEVEN COLLINS

FACULTY OF COMPUTING & MATHEMATICAL SCIENCES

DUBLIN UNIVERSITY COLLEGE

DATE: 31st JANUARY, 1997.

TABLE OF CONTENTS.

ABSTRACT

This paper sets about optimizing the process whereby a 2D texture plane is mapped onto a 3-D quadrilateral planar surface, with a view to achieving enhanced realism of real-time graphical applications such as virtual reality systems.

Each of the possible texturing algorithms are developed[1] and their performance determined[2] by measuring the time taken to render five thousand randomly generated three dimensional quadrilaterals. Included in these algorithms is a previously unpublished algorithm known as the *"Hyperbolic Algorithm"*[3] and a newly developed algorithm by myself, which I have called the *"Adaptive Scanline Sub-division Algorithm"*.

The later algorithm had two possible implementations, one of which was shown to offer excellent image quality, while offering the advantage of increased performance over both scan line sub-division and true mapping. My implementations of hyperbolic mapping however gave both inferior image quality and performance to true mapping as did the second implementation of my adaptive scanline algorithm.

The results obtained remove much of the here say and mystery, that have surrounded the use of perspectively correct texture mapping algorithms in real time, and allow us to draw numerous interesting facts from the performance tests.

Among these is the fact that scanline subdivision only offers superior performance over true mapping when the pixels between the divides is set to eight or more. Area sub-division is pointless beyond level one and should only be implemented with linear and quadratic approximations[4]. In addition, polynomial approximations of an order of four or more are shown to be slower than true mapping. Finally utilizing triangles as a fundamental rendering primitive has a performance disadvantage over quadrilaterals.

The results in this paper provide the virtual reality programmer with solid empirical data, that enables him/her to select the correct algorithm for a desired image quality and a given level of performance.

[1] The following algorithms are considered; true, hyperbolic, linear, quadratic, cubic, Taylor, area sub-division, scanline sub-division, adaptive scanline sub-division. The Constant *Z* algorithm is not considered as it does not allow randomly oriented polygons.

[2] Each algorithm was optimized in so far as possible by the author, however hardware or assembler optimizations were not considered. In addition results are quoted in msecs per 80x80 pixel quadrilateral, dispensing with ambiguous metrics such as pixels per second or polygons per second. Such metrics either disregard the set up overhead per polygon or indeed the size of each polygon. We do however rely on subjective means in determining image quality.

[3] This algorithm was however, published on the internet by Mr Jan Vondrak.

[4] Only linear, quadratic and cubic approximations were tested.

1. INTRODUCTION

1.1 INTRODUCTION TO TEXTURE MAPPING

Texture mapping is about calculating the geometric correspondence between points in texture space or *uv* space and object space.

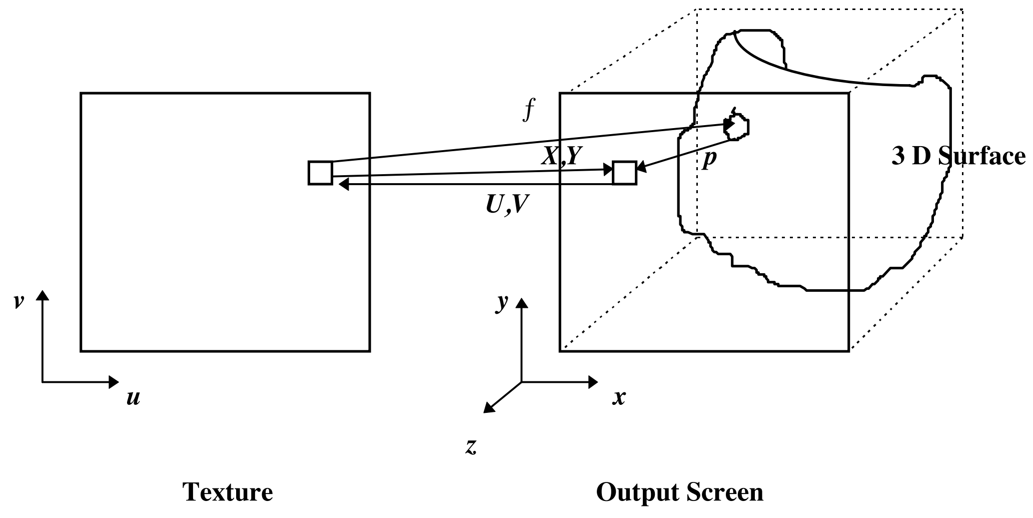


Diagram of functions from *u,v* to x_o,y_o,z_o (object) to x_s,y_s (screen)

In order to determine this correspondence we parameterize texture space using *u, v* co-ordinates, and the object space, using the homogenous co-ordinate system. We then establish the true texture co-ordinates at the quadrilateral vertices and decide on a means of interpolating these values as we move down and across the quadrilateral on screen. We could derive a function that determines the texture co-ordinates from screen pixel co-ordinates, however, it is shown that this involves the solving of a quadratic equation for each texture co-ordinate at each pixel value[5] making it unsuitable for real time applications.

1.2 THE NATURE OF THE PROBLEM

We wish to optimize the process whereby a 2D texture plane is mapped onto a 3-D quadrilateral planar surface, with a view to achieving enhanced realism of real-time graphical applications such as virtual reality systems.

When mapping a 2-D texture plane onto a 3-D quadrilateral surface and then projecting the surface onto the 2-D screen display, we have numerous choices as to the nature of the mapping or indeed how we interpolate the texture co-ordinates across the

[5] Refer to HOUR83.

quadrilateral. We shall see that in order to obtain a perfect image quality we carry out a true mapping which involves two divisions per pixel[6]. Other mappings, such as an affine mapping are only approximating algorithms, but will have lower computational overhead[7]. Such mappings will give a lower image quality[8], but may be all that is required in a given set of circumstances. In general, the more sophisticated the approximating algorithm, the higher the image quality and indeed the higher the associated overhead. Indeed it is important that any such approximating algorithm gives a performance at least as good as a true perspective mapper.

This paper examines the performance of true mapping, relative to the various approximating techniques and attempts to establish which algorithms are worth implementing. Tests were also performed to examine the relative performances of mapping two triangles as oppose to a quadrilateral.

Each algorithm is optimized in so far as possible by the author, however special purpose hardware, assembler and indeed certain optimization techniques such as fixed point mathematics are not considered.

[6] One divide per pixel is possible, however two multiplications per pixel are also required.

[7] An affine mapping has two additions per pixel.

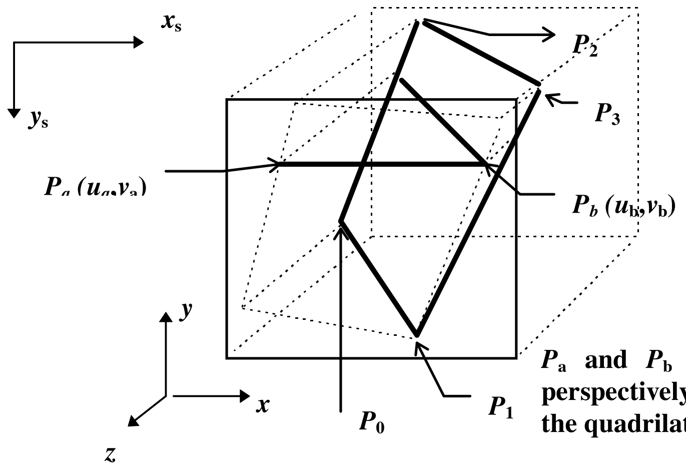
[8] Refer to Colour Plates V and VIII.

2. SCANLINE ALGORITHMS FOR PERSPECTIVE TEXTURE MAPPING

2.1 TRUE PERSPECTIVE TEXTURE MAPPING

2.1.1 True Perspective Mapping With Divides.

Examining the geometry of texture mapped polygon.



P_a and P_b are points in screen space, perspectively projected from the edges of the quadrilateral.

If we examine the texture co-ordinates along the scan line from P_a to P_b we can determine that they do not vary in a linear manner[9] with respect to x_s.(similarly for a vertical scanline with respect to y_s). It can however be determined that u/z and v/z or indeed, u/w and v/w are linear with respect to the screen co-ordinates. Thus, if we interpolate u/w and v/w across a scan line with respect to x_s (similarly for a vertical scanline with respect to y_s) and then divide by $1/w$ we can determine the true texture co-ordinates. Thus, for each pixel we will need to carry out two divisions per pixel. We will also have other calculations to carry out in order to interpolate 1/w and set up computations for each scan line and indeed each quadrilateral.

We could also texture map a quadrilateral by sub-dividing it into two triangles and developing an algorithm to map each triangle in turn. Results however indicate that this actually increases the associated overhead as we now have a larger set up overhead.

[9] They fail to take the form $y = mx + c$, but take on a curvilinear shape.

2.1.2 Hyperbolic Texture Mapping.

We have seen that true mapping utilzes expensive divides. In my search for algorithms that would increase the performance of perspective texture mapping, I came across a new algorithm for texture mapping. This algorithm was first brought to the attention of the world by Mr Jan Vondrak in an article posted on the internet. I however, did not hear about this algorithm until I visited Mr Mark Feldman's web site[10]. The algorithm is so named as it is essentially a development of Bresenham's algorithm for elliptical shapes[11]. The algorithm is in fact a true perspective mapper without the per pixel divides. It may therefore provide us with a degree of performance enhancement while maintaining the image quality of true mapping.

The details are as follows;

From HECK89 it is shown that;

$$u = \frac{a(x - x_a) + b}{c(x - x_a) + d}$$

Cross multiplying and bringing everything to one side results in;

$$p(u,x) = a(x - x_a) + b - u(c(x - x_a) + d) = 0$$

Similarly for v;

$$q(v,x) = e(x - x_a) + f - v(g(x - x_a) + h) = 0$$

These are functions that equal zero[12] if the values of the texture co-ordinates for a given screen co-ordinate represent their corresponding true values. Thus, if we were to start off on one side of the quadrilateral, (u_a and v_a as before) knowing the true values of u and v, we should have $p(u,x)$ and $q(v,x)$ equal to zero. If we then increment x by one and note the effect on $p(u,x)$ and $q(v,x)$ we should be able to determine the increment or decrement in u and v to bring these functions back to zero.

This is easy enough to implement. However, when the image is far away we need to have large step values for the texture co-ordinates, and as we move nearer to the viewer we need to reduce the step values, otherwise performance suffers severely.

[10] http://www.netcom.com/~pcgpe/tmap.html

[11] I myself am unconvinced of the reasoning behind the name.

[12] They will be close to zero, allowing for integer increments in x, u and v.

2.2 APPROXIMATING PERSPECTIVE TEXTURE MAPPING

As can be seen from the previous discussion true perspective texture mapping requires a high degree of computational overhead, regardless of any sophisticated scan line algorithm we may develop. Such computational expense may in some cases only add marginal realism to an image that could be rendered using an approximating algorithm. In the context of real time applications it is therefore worthwhile to examine approximating algorithms, particularly those that are easily optimised in a scan line format, and those that are computationaly less expensive than true perspective mapping[13]. We will consider the following approximating algorithms;

- Linear Interpolation.
- Quadratic Interpolation.
- Cubic Interpolation.
- Taylor Series.
- Area Sub-Division.
- Scanline Sub-Division.
- Adaptive Scanline Sub-Division.

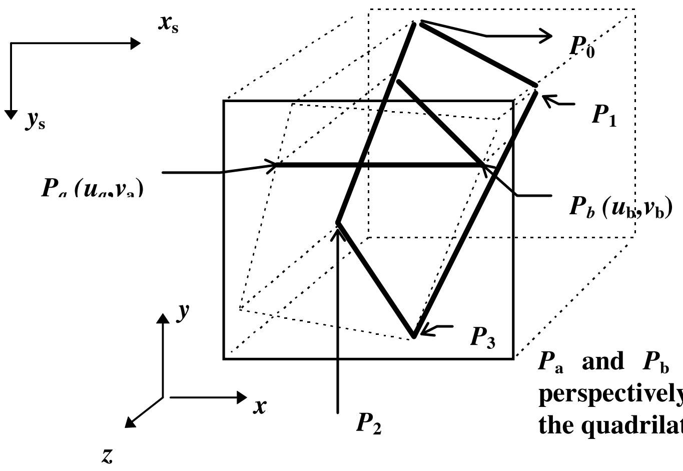
The first four algorithms are essentially polynomial approximations. We will show that higher order polynomials give good image quality, but lack performance. The other algorithms represent good alternatives to polynomial interpolation, with scan line sub-division and adaptive scanline sub-division showing the optimum trade off, of performance versus image quality.
The above polynomial approximations do however lend themselves easily to being incorporated into a scan line algorithm and have previously been discussed in WOLB90. WOLB90 also discussed the area sub-division algorithm. LIEN87 discusses the Taylor series method briefly, while Michael Abrash[14], and HECKER95d. mention the scan line sub-division algorithm as implemented in a computer game engine. HECK91b was the originator of this algorithm however, where it is referred to as piece wise linear approximation. The adaptive scanline sub-division algorithm is a development of this later algorithm, and to the authors knowledge, this paper is the first publication of any such algorithm.

[13] In general the more complex the approximation is, the more likely it is to have a similar computational overhead to true perspective mapping.
[14] *ftp://ftp.idsoftware.com/mikeab/cgdctalk.doc*

2.2.1 Linear Interpolation

This method of texture mapping is similar to the method described under true perspective mapping except the *u* and *v* co-ordinates are interpolated across the polygon with respect to x_s and y_s, rather than with respect to $1/w$. This does away with the need for two divides per pixel.

As before, we determine the *u*,*v* values at the vertices. From these we derive the *u*, *v* co-ordinates along the edges by simple interpolation in x_s and y_s.



P_a and P_b are points in screen space, perspectively projected from the edges of the quadrilateral.

Thus, knowing the values of *u* and *v* at points P_a and P_b we simply linearly interpolate between u_a and u_b as well as v_a and v_b with respect to x_s (similarly for a vertical scanline with respect to y_s). We will thus have to carry out two additions per pixel rather than two divisions.

2.2.2 Quadratic Interpolation

As stated previously, it can be shown that the texture co-ordinates vary across a scan line in a curvilinear manner, similar to a quadratic curve. As such we can interpolate the co-ordinates across a scanline using quadratic equations.

This will take on the following form;

$$u = a_u x^2 + b_u x + c_u$$
$$v = a_v x^2 + b_v x + c_v$$

where $x = \dfrac{x_x - x_a}{x_b - x_a}$, thus ranging from 0 to 1, spanning the length of the scanline.

Since we have three unknown coefficients for each mapping function, three (*x*, *u*) and (*x*, *v*) pairs must be supplied. The three points we select are the two ends of the scanline and its midpoint. From these the coefficients of the quadratic equations can be determined and a scan line algorithm developed using forward differences or partial differentiation. Resulting images using this algorithm are reasonable and it is shown that it also provides superior performance over true mapping.

2.2.3 Cubic Interpolation

Given the success of quadratic interpolation, it is natural to investigate how much better the results may be with cubic interpolation. A third-degree polynomial mapping function for *u* and *v* has the form;

$$u = a_u x^3 + b_u x^2 + c_u x + d_u$$
$$v = a_v x^3 + b_v x^2 + c_v x + d_v$$

again where *x* is a normalised parameter in the range from 0 to 1[15], spanning the length of the scanline.

This time, since we have four unknown coefficients for each equation, four constraints must be imposed. In this instance we determine the texture co-ordinates at the edges of the scan line as before, and at two further points between the edge points[16]. The two points that I selected are the points that divide the scanline in three. The algorithm is exactly the same as quadratic interpolation, except that an additional coefficient needs to be determined and we utilize three forward differences rather than two. Resulting images show a remarkable improvement on quadratic interpolation, and again the algorithm does offer superior performance over true mapping.

2.2.4 Taylor Series

Obviously as we increase the order of the polynomial the better the approximation. However increasing the order of the polynomial will also increase the computational overhead and indeed the difficulty in arriving at the polynomial coefficients. The use of a Taylor expansion will simplify such derivations and may increase computational efficiency.

A Taylor expansion of the function *f* around the point x is;

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!} f''(x) + \frac{h^3}{3!} f'''(x) + \ldots\ldots\ldots$$

---

[15] $x = \frac{x - x_a}{x_b - x_a}$

[16] WOLB90 uses edges constraints and first derivative constraints (Hermite Interpolation). Although it would appear to be computationally more efficient than the cubic interpolating algorithm suggested above it resulted in images approaching the quality of a quadratic interpolation.

If we disregard derivatives of an order greater than three and step the function in increments of 1 rather than h then;

$$f(x+1) = f(x) + f'(x) + \frac{1}{2}f''(x) + \frac{1}{6}f'''(x)$$

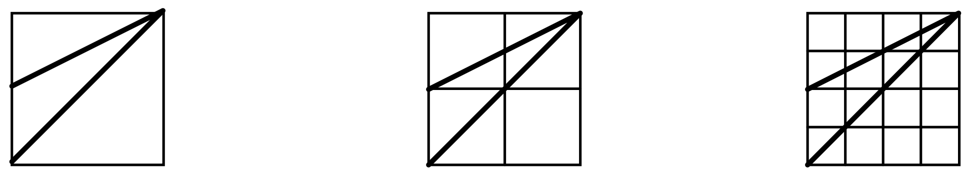
Adapting the above to a function of u;

$$u_{x+1} = u_x + u'_x + \frac{1}{2}u''_x + \frac{1}{6}u'''_x$$

We then carry out a Taylor expansion of the cubic approximation which simplifies down to the set of forward differences as used in the cubic approximation. Thus the only way of achieving computational gain over a cubic approximation is to drop the higher order forward differences which will drastically effect image quality. We therefore, conclude that this algorithm is not worth implementing, but does facilitate the development if a scanline algorithm.

2.2.5 Area Sub Division

If the quadrilateral that we wish to texture map is at a constant z, or parallel to the viewing plane, the texture co-ordinates will be linear with respect to the screen co-ordinates. Indeed, if we have only a minor variation in the z co-ordinate along the quadrilateral, a linear map will give very good image quality, while giving maximum performance gain. Bearing this in mind, if we were to divide the quadrilateral into a series of smaller quadrilaterals, in which the difference in z is small, we could use a linear approximating algorithm and generate images that would be of a quality approaching that of a true perspectively mapped image. This would have to be done such that the level of sub-division did not make it computationally more expensive than true perspective mapping.



(a) **(b)** **(c)**

(a) Shows image before sub division.
(b) Shows image at first level of subdivision.
(c) Shows image at second level of subdivision.

At each level of sub-division the true texture co-ordinates would need to be calculated for each of the sub-quadrilateral vertices. We could then linearly interpolate these values between the vertices as before[17] or interpolate by some other means.

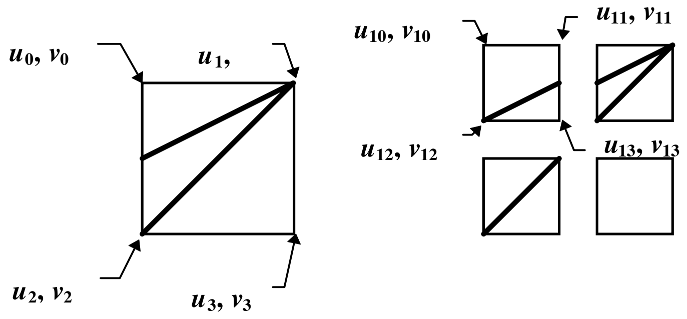


Diagram showing the texture values at the vertices of a quadrilateral and a sub-quadrilateral at level one.

An efficient means of carrying out this sub-division was developed and it was shown that performance gains were apparent over true mapping for one level of sub-division, however further levels of sub-division made true mapping more attractive from a performance view. The use of parallel processing would greatly enhance the performance of this algorithm[18].

2.2.6 Scanline Sub-Division

The scanline sub-division algorithm calculates intermittent true values of texture co-ordinates along a scanline, while linearly interpolating the values between these true values. Thus the resulting performance will be somewhere in between that of a linear approximating algorithm and that of a true perspective mapper[19]. By setting the frequency (pixels between divisions) at which true texture co-ordinates are determined we can vary the performance of the mapper within these limits[20].

[17] WOLB90 demonstrated the use of sub-division using linear, quadratic and cubic interpolation. He noted that less sub-division was required for higher order interpolation functions in order to obtain an image of a quality approaching that of a true perspective mapping. It is worth noting that we could easily use a different approximating algorithm to determine the values of the texture co-ordinates between the vertices of each sub-quadrilateral.

[18] Refer to GILL89.

[19] Because of additional overhead it will in fact be slower. The scanline sub-division algorithm had a time of 30.03 msecs per quadrilateral at one pixel between the divides. This would be equivalent to true mapping which was timed at 18.76 msecs. Likewise, the scanline sub-division algorithm was timed at 16.41 msecs for a pixel number of thirty two between divides. This would be close to linear mapping which was timed at 14.83 msecs per quadrilateral. Again the scanline sub-division is slower.

[20] Michael Abrash discusses this briefly at *ftp://ftp.idsoftware.com/mikeab/cgdctalk.doc*. More detail can be found at *http://www.netcom.com/~pcgpe/tmap.html.*

This algorithm gives excellent results[21] and is quicker than true mapping when the pixels between the divides is set to eight or greater[22]. Indeed, with the number of pixels between divides set to eight, there was little perceptible difference in image quality between true perspective mapping and scanline sub-division. With a pixel block of sixteen the image quality is still very high, while at thirty two it is beginning to degrade to an unacceptable level.

2.2.7 Adaptive Scanline Subdivision.

The previously described algorithm can be speeded up if we were able to vary the number of pixels between divides, according to either how near or far the object was from the viewer, or the difference in z co-ordinates at the quadrilateral edges. This can be done on a per scan line basis, i.e. the number of pixels between divides is determined for each scan line, or it can be done as we move across the scan line.

The former method simply would carry out a test to determine the change in z from edge to edge across a scan line and arrive at a predetermined number of pixels between the divides. The distance a quadrilateral is from the viewer can also be incorporated into the logic, with additional overhead. The method does however show excellent image quality, while offering increased performance over true mapping. It also offers increased performance over scan-line sub-division, when the resulting image quality is taken into account.

The later method involves the recalculation of the pixels between the divides as we move across the scan line. At points further away from the viewer we would require a smaller number of pixels between the divides, where as at z values closer to the viewer, larger values would suffice. We can do this by determining the difference between the true and linear texture co-ordinates as we move across a scan line. If the difference between the two sets of values is large we lower the pixels between the divides, and if the difference is small we can higher the number of pixels between the divides. This algorithm did not give enhanced performance over true mapping, as it was necessary to set an upper limit on the pixel number between the divides, as du and dv needed to be kept constant across a scan line.

[21] Refer to Colour Plate XXI.

[22] Refer to Performance Comparisons section.

3. ALGORITHM PERFORMANCE COMPARISONS

3.1 METHOD

The following test results were obtained on a Pentium PC with a 100 MHz processor, 16 Mbytes of RAM. All code was written in C++ and compiled using a Borland C++ 4.5 compiler, with optimisations set for speed.

The algorithms used have been previously described, however in order to get valid timings for the execution of each algorithm it was necessary to generate 1,000[23] randomly orientated quadrilaterals and texture map each one. The total time taken to do this was measured and divided by 1,000 to get the average execution time per quadrilateral. This was done five times for each algorithm.

The code was executed (and compiled for) under DOS as Windows added between 7 and 12 msecs per quadrilateral, with much larger standard deviations.

The vector co-ordinates of each vertex were limited to ensure the quadrilateral was clipped or limited to screen co-ordinates. Such a method of clipping resulted in the appearance of some unclipped quadrilaterals (this is where sections of a quadrilateral appear on several edges on screen in non contiguous sections). This occurred on a sporadic basis only and would not invalidate the results.

In addition at high levels of sub-division the texture rendering routine would crash. This was as a result of dy being calculated to be zero. A fix to each rendering routine would have added to the program logic (and therefore the associated overhead) and was therefore not included in the implementation of the algorithms. This problem was overcome in the performance tests by simply reducing the sample size.

[23] For area sub-division it was necessary to reduce the number of tests to 300 for level 2 and, 25 for level 3 as invalid quadrilaterals were being produced. This however does not invalidate my conclusions.

3.2 RESULTS

	Average per time per 80x80 pixel quadrilateral (msecs)					
	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Average**
True Mappings						
True Quadrilateral	18.73	18.73	18.78	18.78	18.78	**18.76**
True Triangle	21.97	21.97	21.97	21.97	21.97	**21.97**
Hyperbolic (Slow)	36.74	36.80	36.80	36.80	36.80	**36.79**
Hyperbolic (Fast)	24.50	24.55	24.50	24.49	24.49	**24.51**
Approximating Algorithms						
Linear	14.82	14.83	14.83	14.83	14.83	**14.83**
Linear (True Edges)	15.10	15.16	15.15	15.16	15.16	**15.15**
Quadratic	16.47	16.48	16.48	16.47	16.48	**16.48**
Cubic	17.90	17.96	17.96	17.97	17.96	**17.95**
Scan line Sub-division (dynamic)						
1 pixel between divides	29.98	30.10	29.99	30.05	30.04	**30.03**
2 pixels between divides	22.91	22.90	22.85	22.91	22.91	**22.90**
4 pixels between divides	19.44	19.44	19.44	19.45	19.44	**19.44**
8 pixels between divides	17.74	17.74	17.69	17.74	17.74	**17.73**
16 pixels between divides	16.86	16.86	16.86	16.87	16.86	**16.86**
32 pixels between divides	16.37	16.42	16.42	16.43	16.42	**16.41**
Adaptive Scan line Sub-division (static)						
Max pixel block 8 pixels	28.50	28.51	28.45	28.51	28.46	**28.49**
Max pixel block 16 pixels	27.74	27.74	27.74	27.74	27.73	**27.74**
Adaptive Scan line Sub-division						
Pixel blocks 2,4,8,16	18.73	18.73	18.73	18.73	18.73	**18.73**
Pixel blocks 4,8,16,32	17.41	17.41	17.36	17.35	17.36	**17.38**
Pixel blocks 8,16,32,64	16.70	16.70	16.64	16.70	16.64	**16.68**
Adaptive Scan line Sub-division, Z						
Pixel blocks 2,4,8,16	19.55	19.55	19.56	19.55	19.56	**19.55**
Pixel blocks 4,8,16,32	17.79	17.80	17.80	17.80	17.79	**17.80**
Pixel blocks 8,16,32,64	16.91	16.92	16.92	16.92	16.92	**16.92**
Area Sub-division, Linear						
Level 1	16.04	16.09	16.09	16.10	16.04	**16.07**
*Level 2**	20.17	20.33	20.33	20.33	20.33	**20.30**
*Level 3***	30.80	30.40	30.80	30.80	30.80	**30.72**
Area Sub-division, Quadratic						
Level 1	18.46	18.40	18.46	18.45	18.46	**18.45**
*Level 2**	23.96	24.17	24.17	24.17	24.17	**24.13**
*Level 3***	37.20	39.60	39.60	39.60	39.60	**39.12**
Area Sub-division, Cubic						
Level 1	20.27	20.32	20.32	20.32	20.36	**20.32**
*Level 2**	26.90	26.73	26.90	26.93	26.93	**26.88**
*Level 3***	44.00	43.60	44.00	44.00	44.00	**43.92**

*Sample size decreased to 300
**Sample size decreased to 25

4. CONCLUSION

Overall, the performance of our true and approximating texture mapping algorithms was quite good, as we managed to reduce the rendering time to below the required level for good quality real time graphics. We would not however, be able to draw many polygons on screen without further speeding up the algorithm. Further enhancements could easily be achieved through the use of enhanced hardware[24] or indeed the use of assembler code[25]. The results also allow us to determine which approximating algorithms are worth while implementing, having specified a proper means of comparison and therefore clear up the mystic that has previously surrounded perspective texture mapping algorithms.

The results show that sub-dividing a quadrilateral into triangles, results in increased overhead and results in a slower texture mapping speed. This demonstrates that a simple change to an algorithm can generate higher performance levels, without having to resort to assembler or hardware. This is not to say that using triangles does not have any advantages. Texture mapping triangles is easily optimized using hardware and as all triangles are planar they do not require a planarity test. Consequently, the use of edge classes for triangles is also not required, as an algorithm can be derived based solely on the vertices[26]. Indeed, most multi-sided objects are easily broken down into a series of triangles rather than quadrilaterals. These points are however outweighed by the superior performance of quadrilaterals over triangles, and bearing in mind that quadrilaterals are also easily optimized in hardware. Indeed, most virtual reality systems will rely heavily on texture mapped quadrilaterals and most objects are also easily broken down into a series of quadrilaterals. Such systems will not generate non planar quadrilaterals and will therefore not require a test of planarity. In addition, mapping a quadrilateral using an approximating technique will not result in large diagonal distortions as with triangle sub-division[27].

Having discovered a new texture mapping algorithm, namely Hyperbolic texture mapping, I was somewhat disappointed to discover it was slower than true mapping in addition to generating an inferior image. I did not however implement an exact copy of the original algorithm as per the author, as I catered for all orientations of the quadrilaterals. Performance however may be enhanced through either restricting the

[24] Refer to COOK87, KIRK90 & OKA87 for hardware architectures specifically designed for texture mapping. GUAN94 details hardware used for 3-D textures.

[25] Refer to HECKER95a - HECKER96 or *http://www.gamers.org/dEngine/quote/papers/checker_texmap.html.*

[26] Such an algorithm would enhance the performance of triangular mapping, however my results indicate that quadrilateral mapping is still quicker taking this into account.

[27] Refer to the comparison of Colour Plates XXIX and XXX. The later plate was mapped using triangles. Distortion is noted across the diagonal.

vertices of the polygons that are generated, or through an alteration in the algorithm. A speed increase may also be achieved through the use of fixed point numbers, as the division and multiplication necessary to increment or decrement the step values could then be achieved using a shift operation. A corresponding increase however would also be noted in true texture mapping, if fixed point numbers were used. Such an increase in performance would be marginal, given that, in recent years the gap between the speed of floating point calculations relative to integer calculations has been drastically reduced[28]. Given these two potential performance enhancements, I am somewhat reluctant to disregard hyperbolic texture mapping without further work.

Of all the approximating algorithms the scan line subdivision algorithm offered both good image quality and a marginal increases in performance over true mapping. Such increase in performance was only noted when the pixels between the divides were set to eight or more. Bearing this in mind I developed two algorithms which were essentially a development of the scan line sub-division algorithm. As these new algorithms essentially determined the pixels between divides in an intelligent manner I called them *"Adaptive Scanline Sub-division."* One of these algorithms recalculates the pixels between divides as we move across a scan line, while the other recalculates the pixels between divides on a per scan line basis. The later algorithm gives superior performance to scan line sub division and true mapping despite its increased overhead per scanline. The former algorithm performs badly in the test, but would offer excellent quality images at good performance levels for quadrilaterals that lie obliquely to the screen.

The results expose the weakness of using area sub-division with approximating algorithms as no performance enhancements over true mapping were noted beyond level one[29]. This was only the case for linear and quadratic approximations, as sub-dividing a cubic approximation to level one was actually slower than true mapping.

Finally, the use of polynomial approximations beyond the power of three would not offer us any performance gain over true mapping, even using a Taylor approximation.

[28] Some chip manufacturers quote instruction timings for floating point calculations lower than integer calculations.

[29] Parallel hardware configurations were not considered. Refer to GILL89.

5. REFERENCES

COOK87 Cook, R. L., Carpenter, L. and Catmull, E., *"The REYES Image Rendering Architecture."* Proceedings of SIGGRAPH '87 (Anaheim, California, July 27-31, 1987). In Computer Graphics, Volume 21(4), July 1987, ACM SIGGRAPH, New York, p95-102.

FOLEY90 Foley, J., van Dam, A., Feiner, S. and Hughes, J., *"Computer Graphics, Principles and Practice."* 2nd edition, Addison Wesley, 1990.

GANG82 Gangner, M., Perny D. and Coueignoux, P., *"Perspective Mapping of Planar Textures."* Proceedings of EUROGRAPHICS '82 (North Holland, Amsterdam, 1982), p57-71.

GILL89 Gillies, Duncan. and Burger, Peter. *"Parallel Texture Mapping for Low Cost Animation Systems."* At Computer Graphics 89, Blenheim On-line Publications, Pinner, Middx, UK, 1989, p367-373.

GUAN94 Guan, Sheng-Yih. and Lipes, Richard. G., *"Innovative Volume Rendering Using 3D Texture Mapping."* Proceedings of the SPIE - The International Society for Optical Engineering, Volume 2164 (Newport Beach, CA, USA, 1994) p382-392.

HECK83 Heckbert, P. S., "Texture Mapping Polygons in Perspective." Tech Memo No. 13, NYIT Computer Graphics Lab. April 1983.

HECK86 Heckbert, P. S., *"Survey of Texture Mapping."* IEEE Computer Graphics and Applications, Volume 6(11), November 1986, p56-67.

HECK89 Heckbert, P. S., *"Fundamentals of Texture Mapping and Image Warping"* Masters Thesis, Dept of EECS, University of California at Berkeley, Technical Report No. UCB/CSD 89/516, June 1989.

HECK91b Heckbert, P. S. and Moreton, Henry P., *"Interpolation for Polygon Texture Mapping and Shading."* In Rogers, David F. and Earnshaw, Rae A. editors, *"State of the Art in Computer Graphics: Visualization and Modelling."* p101-111, Springer-Verlag, 1991.

HECKER95a Hecker, Chris., *“Perspective Texture Mapping Part I: Foundations.”* Game Developer, Volume 2, No. 2, April/May 1995, p16-25.

HECKER95b Hecker, Chris., *“Perspective Texture Mapping Part II: Rasterization.”* Game Developer, Volume 2, No. 3, June/July 1995, p18-26.

HECKER95c Hecker, Chris., *“Perspective Texture Mapping Part III: End Points & Mapping.”* Game Developer, Volume 2, No. 4, August/September 1995, p17-24.

HECKER95d Hecker, Chris., *“Perspective Texture Mapping Part IV: Approximations.”* Game Developer, Volume 2, No. 6, December/January 1995, p19-25.

HECKER96 Hecker, Chris., *“Perspective Texture Mapping Part V: It’s About Time.”* Game Developer, Volume 3, No. 2, April/May 1996, p25-33.

HOUR83 Hourcade, J. C. and Nicolas, A. *“Inverse Perspective Mapping in Scanline Order onto Non-Planar Quadrilaterals.”* EUROGRAPHICS 83, p309-319.

KIRK90 Kirk, David. and Voorhies, Douglas., *“The Rendering Architecture of DN1000VS”,* Proceedings of SIGGRAPH ‘90. In Computer Graphics, Volume 24(4), August 1990, ACM SIGGRAPH, New York, p299-307.

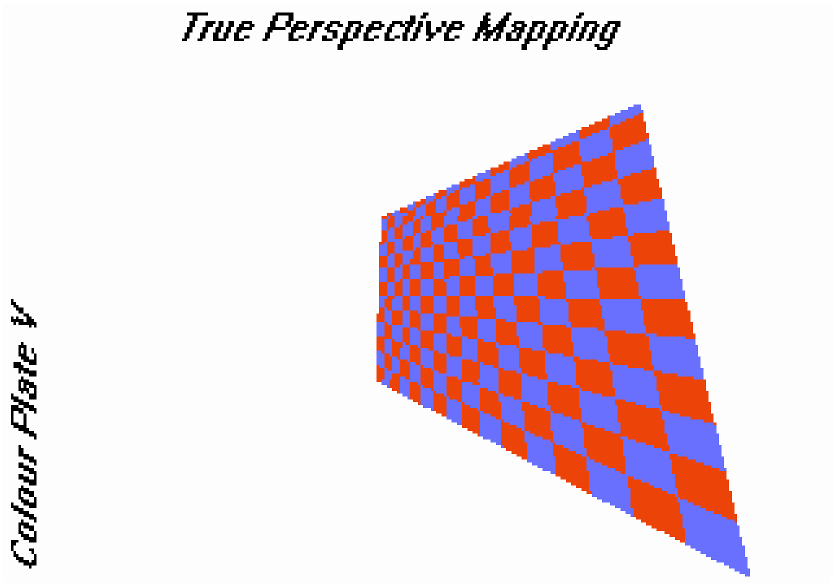
LIEN87 Lien, S. L., Shantz, M., and Pratt, V. "*Adaptive Forward Differencing for Rendering Curves and Surfaces,*" Proceedings of SIGGRAPH ‘87 (Anaheim, California, July 27-31, 1987). In Computer Graphics, Volume 21(4), July 1987, ACM SIGGRAPH, New York, p. 111-118.

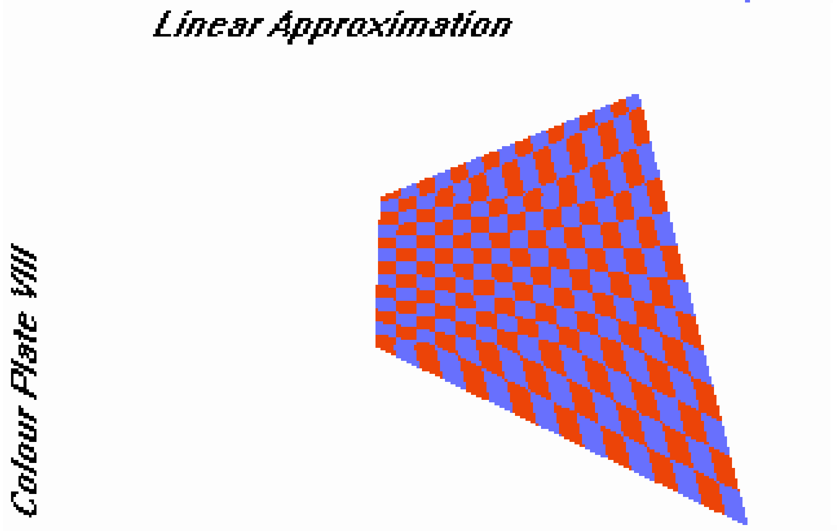
OKA87 Oka, M., K. Tsutsui, A. Ohba, Y. Kurauchi, and T. Tagao*,* "*Real-Time Manipulation of Texture-Mapped Surfaces,*" Proceedings of SIGGRAPH ‘87 (Anaheim, California, July 27-31, 1987). In Computer Graphics, Volume 21(4), July 1987, ACM SIGGRAPH, New York, p181-188.

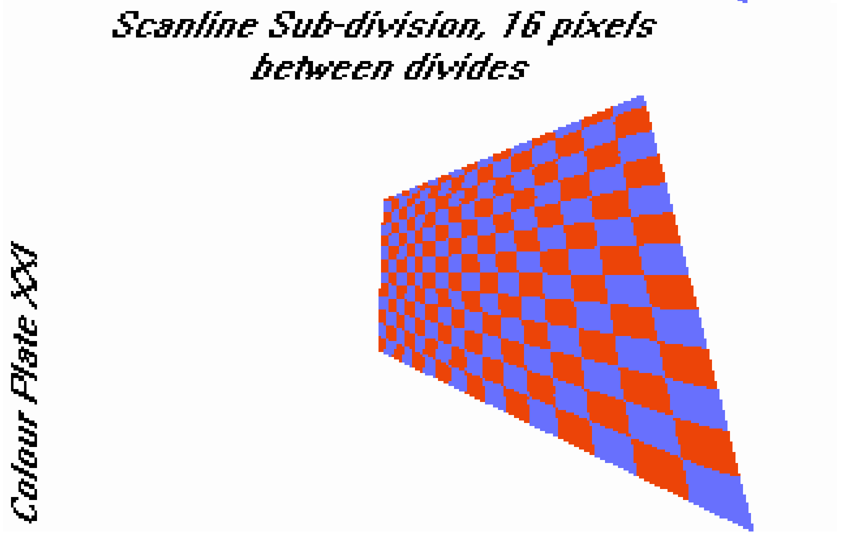
SMITH80, Aly Ray Smith, *“Incremental Rendering of Textures in Perspective.”* SIGGRAPH 80: Animation Graphics Seminar Notes, July 1980.

WOLB90 Wolberg, G. *“Digital Image Warping.”* IEEE Computer Society Press, 1990.

6. COLOUR PLATES


True Perspective Mapping
Colour Plate V


Linear Approximation
Colour Plate VIII


Scanline Sub-division, 16 pixels between divides
Colour Plate XXI