A VLSI Chip for Ray Tracing Bicubic Patches

R. W. Pulleyblank* and J. Kapenga**

*Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA, 94304, USA
**Western Michigan University, Department of Computer Science, Kalamazoo, MI, USA

ABSTRACT

A VLSI chip for ray tracing bicubic patches in Bezier form is explored. The purpose of the chip is to calculate the intersection point of a ray with the bicubic patch to a specified level of accuracy, returning the location of the intersection on the patch and on the ray. This is done by computing the intersection of the ray with a bounding volume of the patch and repeatedly subdividing the patch until the bounding volume of subpatches hit by the ray is smaller than the accuracy requirement. There are two operating modes, one in which only the nearest intersection is found and another in which all intersections are found. This algorithm correctly handles rays tangentially intersecting a planar patch and ray intersections at a silhouette edge of the patch. Estimates indicate that such a chip could be implemented in 2 micron NMOS and could compute patch/ray intersections at the rate of one every 15 microseconds for patches that are prescaled and specified to 12 bits fixed point for each of the x, y and z components. A version capable of handling 24 bit patches could compute patch/ray intersections at the rate of one every 140 microseconds. Images drawn using a software version of the algorithm are presented and discussed.

INTRODUCTION

Ray tracing is a powerful technique used to create very realistic images (Whitted, 1980) and to compute properties of solids, such as volume and moments, in solid modeling systems (Roth, 1982). Frequently modeling systems are restricted to objects described by polygons or quadratic surfaces and objects that are compositions of them. Curved surfaces can be described efficiently by bicubic patches (Foley and Van Dam, 1982), which specify points on the surface with a function that is cubic in each of it's two parameters, u and v. These patches are determined by a collection of sixteen (three dimensional) control points. Because these patches can be pieced together while maintaining second order continuity at the boundaries, they are very useful for modeling smooth surfaces.

Bicubic patches are frequently displayed by decomposing them into polygons before further processing or by directly computing ray/patch intersections. Methods for computing these intersections based on root finding (Kajiya, 1982 and Toth, 1985) have numerical stability problems and result in excessive computation times for typical images which may require a hundred million ray/patch intersections. Subdivision techniques for bicubic patches have been used widely for display purposes (Catmull, 1974; Lane, Carpenter and Whitted, 1980; and Clark, 1979) and for processing before beginning a Newton-Raphson search (Toth, 1985 and Sweeney and Bartels, 1986). Whitted (1980) and Lane and Risenfeld (1980) have also used subdivision for direct calculation of ray/patch intersections in software. So far, the long delays due to the computational burden of the ray/patch intersections have restricted the usefulness of bicubic patches in computer graphics and solid modeling systems.

Subdivision of Bezier surfaces is well suited for VLSI implementation because it requires only additions and divisions by two when restricted appropriately. Thus it has been selected to compute ray/patch intersections in what follows. The goal is to reduce the time required for ray/patch intersections to a value that would make it much more attractive to use patches in an interactive environment and to accomplish this with relatively simple hardware.

ALGORITHM

The algorithm relies on the following facts. Bezier curves are easily subdivided into halves (Lane and Risenfeld, 1980 and Lane, Carpenter and Whitted, 1980), requiring only additions and divisions by two of the control points. This basic operation is used repeatedly to subdivide a bicubic patch into quadrants (Lane and Risenfeld, 1980). The control points can be used to form the convex hull of the control points, a volume which completely contains the patch. A much simpler but looser bounding volume can be found by taking the maximum and minimum of the x, y and z components of the control points to form a rectangular bounding box that is aligned with the coordinate axes and it is this method that is actually used in this algorithm. The intersection of a ray with each of these bounding planes is computed and the results examined to determine if the ray intersects the bounding volume.

With these basic ideas in mind, the algorithm can be stated as follows:

To find the intersection of a patch with the ray, the patch is broken into four subpatches, each of whose bounding boxes are computed and tested for an intersection with the ray. If the ray hits the bounding box of a subpatch and the termination conditions are not met, i.e. the patch is not smaller than a specified accuracy requirement and the maximum level of subdivision has not been reached, the subpatch is placed on a stack to be processed further.

This process proceeds in a depth first search for the subpatches whose bounding boxes intersect the ray and which meet the accuracy or maximum subdivision criteria. These *result* patches specify the intersection points. The (u,v) coordinates of the intersection point on the original patch and the parameter t, which specifies the intersection point on the ray, are returned as results.

A conceptual block diagram of the process is shown in Fig. 1.

Roughly one bit of accuracy is obtained for each level of subdivision and the computation is carried out with more bits than the maximum level of subdivision.

It is possible that more than one *result* patch is found for a single intersection. This is the case when the ray intersection is close to the boundary between subpatches or when the ray is parallel to and contained in a planar portion of the patch.

Two approaches may be taken to deal with this situation. The fastest and simplest approach is to report only the *result* patch nearest to the ray origin, i.e. the one with the smallest t, as the answer. If more than one fits this criterion, an arbitrary patch may be selected, or all *result* patches that are *nearest* can be reported, leaving it to the host to either average these *results* or compute the bounding box which contains all of the *results*.

A second approach is to report all *result* subpatches leaving it to the host to group contiguous *results* together to form intersections, and to summarize the resulting intersections by either averaging the *results* or computing the bounding volume of the *results*.

If only the *nearest* intersection is to be found, faster operation may be obtained by discarding all subpatches that are hit by the ray beyond the bounding volume of the current *nearest result* patch, and sorting, nearest to the origin first, those that remain before placing them on the test stack. The *nearest result* patch is updated when a closer one is found. Searching for the *nearest* intersection is well suited to displaying patches but forgoes the possibility of computing all the intersections of a ray and a patch as would be necessary when computing volumes or computing the portion of a ray *on* a surface.

In order to effectively use a restricted number of bits, the patch control points are assumed to be presented to the chip in a coordinate system whose origin is at the minimum bounding box corner. This makes all control points positive numbers. They are also assumed to be scaled so that the largest control point component is less than one and greater than or equal to one half. The ray origin is also presented in this coordinate system and it has been moved to the point where it intersects the bounding box of the original patch. The ray direction vector is normalized to one.

The Bezier form of the bicubic patch was chosen because the control points form a tighter bounding volume than do the B-spline control points. This is not a limitation because patches can be easily converted between the Bezier, B-spline and Hermite forms (Foley and Van Dam, 1982).

VLSI IMPLEMENTATION ISSUES

A bit serial implementation was chosen because of the large number of additions that are required. A parallel computation in x, y and z was chosen because it operates three times as fast as a serial version and there appears to be space enough on the chip to allow it.

A key component of the system is the block that accepts four control points of a Bezier curve and performs the required averaging to produce the control points of the the two halves of the subdivided curve. The block diagram for a circuit that accomplishes this for one component of the control points, x, y or z, is shown in Fig. 2.

The algorithm requires the saving of subpatches whose bounding boxes are hit by the ray and therefore require further subdivision. If the control points of these intermediate patches are to be saved a very large control point memory is required. It is possible, however, to regenerate these intermediate patches from the original while saving only the *path* to the subpatch from the original. This *path* is simply the local (u,v) coordinates (binary fixed point) of the lower left corner of the subpatch on the original patch. As will become apparent as the chip organization is explained, a hybrid of these two approaches, is most attractive.

An intermediate patch may be reproduced for further testing from the original control points by multiple use of a circuit that subdivides a patch and selects one of the resulting quadrants or is bypassed completely. A block diagram of such a circuit is shown in Fig. 3. These circuits can be used in a chain to reproduce any intermediate patch. The resulting patch can then be fed to a subdivide circuit that produces the control points of all of its quadrants, which are in turn, fed to four identical networks that test if the ray *hits* that subpatch. Each of these networks computes the bounding box of its input patch, the intersection points of the ray with the planes of the bounding box, checks the results for an intersection of the ray and the bounding box, and tests if the accuracy criteria are met. The ray *hit* and *accuracy met* information for each quadrant of the test patch is used to decide whether any of these subpatches should be saved for further testing. The *paths* to the patches that require further testing, together with the depth to which they have been divided, are pushed onto a stack for further testing.

If all test patches are produced directly from the original patch as just described, then a subdivide chain that can divide to the maximum depth in a single pass is required. The required maximum subdivision depth is approximately the number of bits of accuracy required so that the chain needs to be very long unless the accuracy required is small. This, of course, requires sizable chip area but also time. The reason is that each subdivide and select circuit has a cascade of six additions, that may produce six carries, which when operating bit serially requires six additional clock cycles for every subdivision stage in the chain. These clock cycles appear as buffer bits between successive patches to be subdivided.

The problem can be alleviated by not always generating a test patch directly from the original patch in one pass but saving the control points at selected intermediate points and then generating the test patch from the saved points. This process we call subdivision in stages. In order to accomplish this, the control point memory is organized as a stack to allow for the storage of intermediate control point arrays and the length of the subdivide chain is Ds=(D-1)/N+1 where D is the maximum depth to which a patch must be subdivided and N is the number of subdivision stages. If a patch has been divided an additional Ds levels from the last set of stored control points and it required further testing, the control points of its immediate parent could be pushed onto the *control point* stack and the path from the these stored control points to it saved. Further refinement would proceed by subdividing these intermediate control points.

With this organization the time penalty is paid only for the additional levels of subdivision from the stored control points. Another benefit is that the test stack memory required is smaller because only the path from the saved control points need be kept on the test stack. Additional patch memory is required for each stage but there is a net savings in area for the proper choice of the number of stages.

The time lost because of the need to provide buffer bits between successive patch subdivisions can be eliminated as far as the rest of the chip is concerned by duplicating the subdivide chain. Thus when buffer bits are being clocked into one subdivide chain, a new computation can begin in a duplicate chain and a steady stream of subdivided patches may be presented to the circuitry that computes bounding boxes.

The system can achieve high throughput by pipelining, but the depth first search requires the result of a computation before a decision on which way to proceed can be made. Efficient utilization of the pipe can be maintained by having several patches on the chip and working on them in rotation. If the time to initiate a computation on each of the patches is greater than the delay, there will be no decrease in throughput. Additional memory for the control points and test stack for each of the patch intersection computations being carried out on the chip is required.

If the chip is seeking only *nearest* intersections, and if the current patch is a *result* , i.e. if it is a *hit* that satisfies the accuracy or maximum subdivision criteria, then a comparison of the newest t value(s) for entering the bounding volume, t in, is made with the t in for the current *nearest result* patch and the current *nearest t in* is updated. Only hits that are closer than the *nearest result* are reported as *hits* in this mode of operation.

The chip, as described so far, carries out intersection calculations for a single ray against a patch, but it can also test the patch against a list of rays if the *compute intersection*, *test for hit* and *compare to nearest* circuits are replicated for each ray. This is a useful addition for primary rays which are highly coherent and for multiple rays used to perform antialiasing.

The overall block diagram of the system described is shown in Fig. 4.

COMMUNICATION WITH THE HOST

The I/O bandwidth requirements for this chip are very small. If L is the number of bits for each component, x, y or z, then roughly L squared serial clock cycles are required to find an intersection and 48*L bits to specify a new patch, 6*L bits for a new ray and 3*L bits to report a result. If it is assumed that patches are kept *on board* the chip and rays are fed in and intersections returned, and it is assumed that the I/O clock runs at one half the serial clock rate, then 2 pins can handle the required bandwidth for L=12 and 1 pin is required for L=24. If a new patch is required in the same time, 8 additional pins are required to handle the bandwidth for L=12 and 4 additional pins are needed for L=24. These numbers are very reasonable, so there is no cause for concern that lack of I/O bandwidth will reduce the effectiveness of the chip.

CHIP SIZE AND PERFORMANCE ESTIMATES

To get a feeling for how realistic it is to implement the circuit in VLSI and to get an idea of its performance, the area and speed were estimated for implementation by a 2 micron NMOS process. The following areas were assumed for the required circuits:

Circuit	Area (square microns)
full adder	900
half adder	450
dynamic shift register bit	300
static shift register bit	375
2:1 multiplexer	225

A serial clock rate of 20 megahertz was assumed and a whole chip was taken to be 36 square millimeters (0.056 square inches). No attempt was made to make a detailed estimate of the area required for data routing and control, although the serial data format simplifies routing and the control is thought to be relatively simple. Instead it was assumed that roughly half the chip would be required for routing, I/O and control.

Rough estimates for the rate at which ray/patch intersections are calculated are provided by multiplying the average time to subdivide a patch once by the ratio of the total number of times the patch must be subdivided to the number of patch intersections found, an experimentally determined number.

Two examples are presented, one that corresponds to a level of accuracy relevant for display purposes, and the other a higher accuracy computation that might be required in a solids modeling context.

Example I

Accuracy suitable for display purposes

L	12	bits per control point component
D	9	subdivisions maximum
N	2	stages of subdivision
Ds	5	subdivisions per stage maximum
S	3	subdivision chains
P	11	simultaneous patch/ray computations
R	4	rays tested against each patch

The average time required to compute a patch/ray intersection is estimated to be 15 microseconds. Forty two percent of a 36 square millimeter chip is required to perform these calculations. In this example, the chip is capable of simultaneously searching for the intersections of the patch with four different rays which could possibly improve the intersection calculation rate to one every 3.7 microseconds.

Example II

Higher accuracy

L	24	bits per control point component
D	21	subdivisions maximum
N	3	stages of subdivision
Ds	8	subdivisions per stage maximum
S	1	subdivision chains
P	6	simultaneous patch/ray computations
R	1	ray tested against each patch

The average time required to compute a patch/ray intersection is estimated to be 140 microseconds. Fifty percent of a 36 square millimeter chip is required.

IMAGES

Identical perspective views of a single bicubic patch drawn with a software version of the algorithm described in section 2 are shown in Fig. 5 to 10. The maximum subdivision level is varied as a parameter and the accuracy requirement is set so that the maximum subdivision level terminates the subdivision process. The pixel intensity is a linear function of the dot product between the surface normal at the intersection point and the ray. An eight bit color lookup table was used.

A 400 by 400 image was generated that essentially spans the viewing window so that roughly one four hundredth (2 to the power -8.5) of the patch projects to one pixel. This means that adjacent rays can be distinguished with roughly 9 levels of subdivision. This is only approximately true because the patch is subdivided uniformly in its parameter space not in real space. The more uniformly spaced the Bezier control points the better the approximation.

The impairments for images with 9 or more levels of subdivision are imperceptible, even on the blowups, except for aliasing errors (jaggies) as can be seen in Fig. 5 and 6. At 7 levels of subdivision, small anomalies are visible in Fig. 7 and Fig. 8, where a slight lighter, darker, lighter, darker, etc. pattern appears parallel to the silhouette edge which is clearly not bicubic behavior but computational inaccuracy. Another impairment is that the color changes no longer appear continuous causing the slightly noticeable *banded* appearance in Fig. 7. This is due to the fact that the minimum size bounding boxes, when projected onto the viewing plane, are significantly larger than individual pixels. An additional inaccuracy due to the same cause is that the straight line edge of the patch in the foreground appears discontinuous where the silhouette edge of the ridge meets the edge of the patch.

As the maximum subdivision level becomes smaller the impairments increase until at 3 levels of subdivision the large bounding boxes shown in Fig. 10 are very clearly distinguishable.

EXTENSIONS

The normal to the surface at the ray intersection point is usually required in computer graphics applications for shading calculations. This information is easily obtained from the control points of the *result* patch calculated by the proposed chip. This follows from the fact that the four *corner* control points in the Bezier form are actually on the patch. The simplest way to extract a normal is to assume that *result* patches are approximately planar, subtract *corner* control points to form vectors in the plane, and take the cross product of the resulting vectors.

The control points of the *result* patch must be known to roughly twice the accuracy required for the intersection alone in order to calculate the normal to the same accuracy as the intersection. Because the VLSI implementation of the subdivide and select chain subdivides patches exactly, it can be easily adapted to produce the corner point difference vectors to the required accuracy with a very modest increase in time. It might be possible to include the cross product calculations on the chip as well. The tradeoffs involved in doing this are the subject of a further investigation.

CONCLUSIONS

The subdivision approach to computing patch/ray intersections is suitable for VLSI implementation. It appears to fit comfortably on a single chip, its performance is not limited by the need for high I/O bandwidth and estimates of its performance are attractive. A software version of the algorithm is being used to generate images in order to gain insights for the proper choice of the chip parameters. Current work is directed towards finding the most effective way to use this chip to implement a powerful graphics or solid/surface modeling environment. This work is focused on making as much use of of these chips in a parallel configuration as possible.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support for this work provided by a grant from STW (The Netherlands) under the Megabit program through the Network Theory Section of the Delft University of Technology. HP Labs support for a sabbatical leave for the first author is greatly appreciated as are the many useful discussions with Ed Deprettere and Rein Nouta on VLSI implementation issues and the system described herein.

REFERENCES

Catmull EE (1974) A subdivision algorithm for computer display of curved surfaces. University of Utah Ph.D. dissertation in Computer Science.

Clark JA (1980) A fast algorithm for rendering parametric surfaces.

Foley JD, Van Dam A (1982) Fundamentals of interactive computer graphics. Addison-Wesley, 514-536.

Kajiya JT (1982) Ray tracing parametric patches. Computer Graphics 16.3: 245-254.

Lane JM, Carpenter LC, Whitted T, Blinn JF (1980) Scan line methods for displaying parametrically defined surfaces. Communications of the ACM 23.1: 468-479.

Lane JM, Risenfeld RF (1980) A theoretical development for the computer generation and display of piecewise polynomial surfaces. IEEE Transactions on Pattern Analysis and Machine Intelligence 2.1: 35-46.

Roth, SD (1982) Ray casting for modeling solids. Computer Graphics and Image Processing 18: 109-144.

Sweeney MAJ, Bartels RH (1986) Ray tracing free-form B-spline surfaces. IEEE Computer Graphics and Applications 6.2: 41-49.

Toth DL (1985) On ray tracing parametric surfaces. Computer Graphics 19.3: 171-179.

Whitted T (1980) An improved illumination model for shaded display. Graphics and Image Processing 23.6: 343-349.

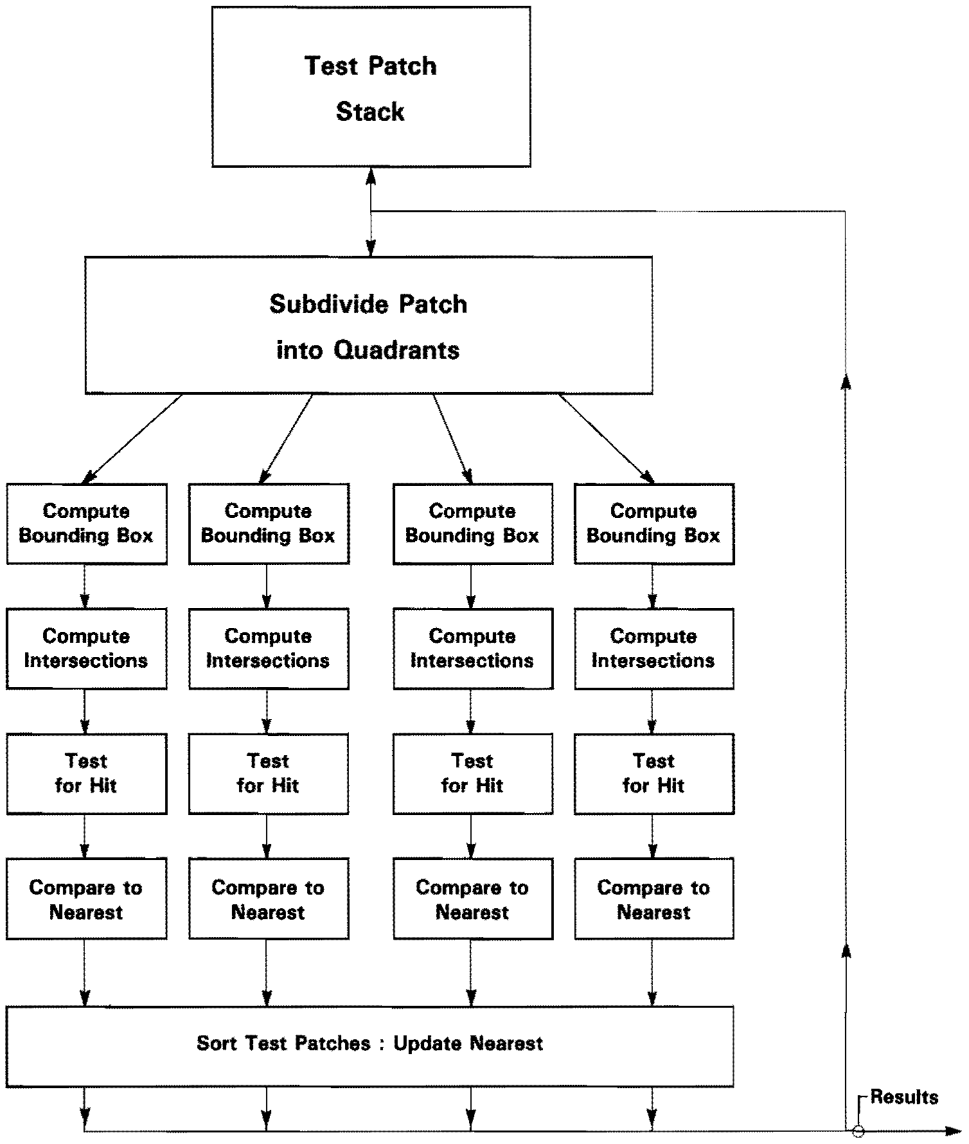
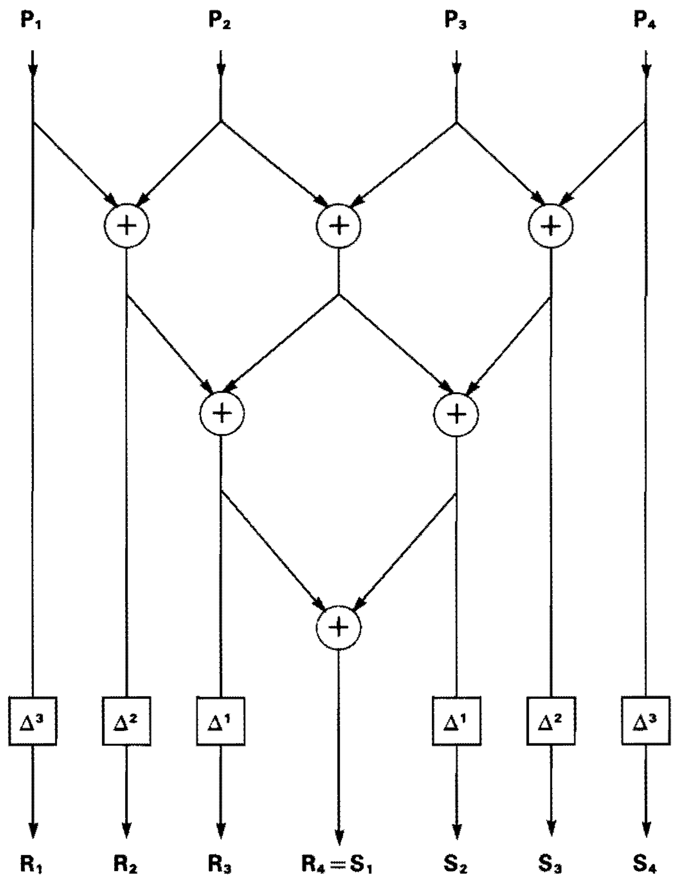


Figure 1

Conceptual Block Diagram



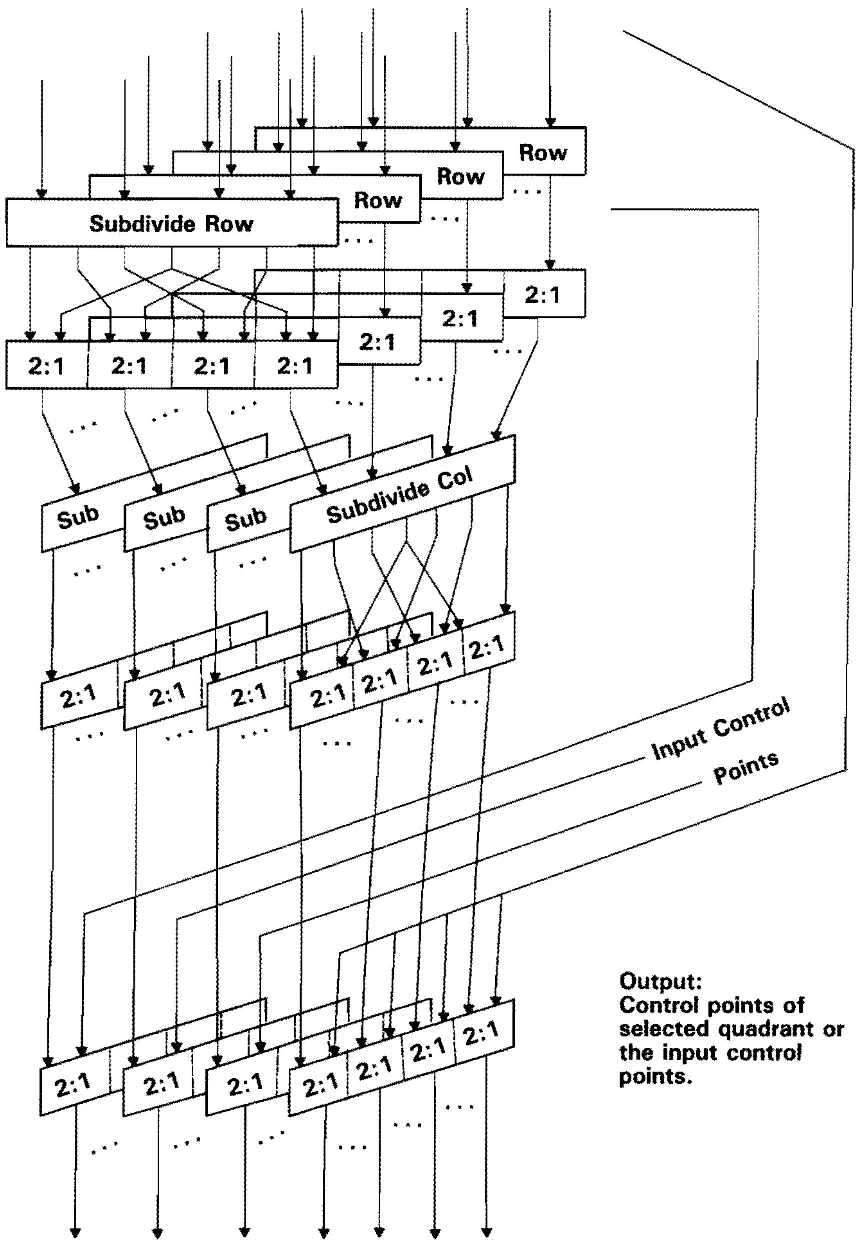
⊕ 1 bit Serial Adder

$\boxed{\Delta^i}$ Delay of i clock cycles

Figure 2

Bit Serial Circuit for Subdivision of Bezier Curve

Input: 16 control points



Output:
Control points of selected quadrant or the input control points.

Figure 3

Subdivide and Select Block (one component)

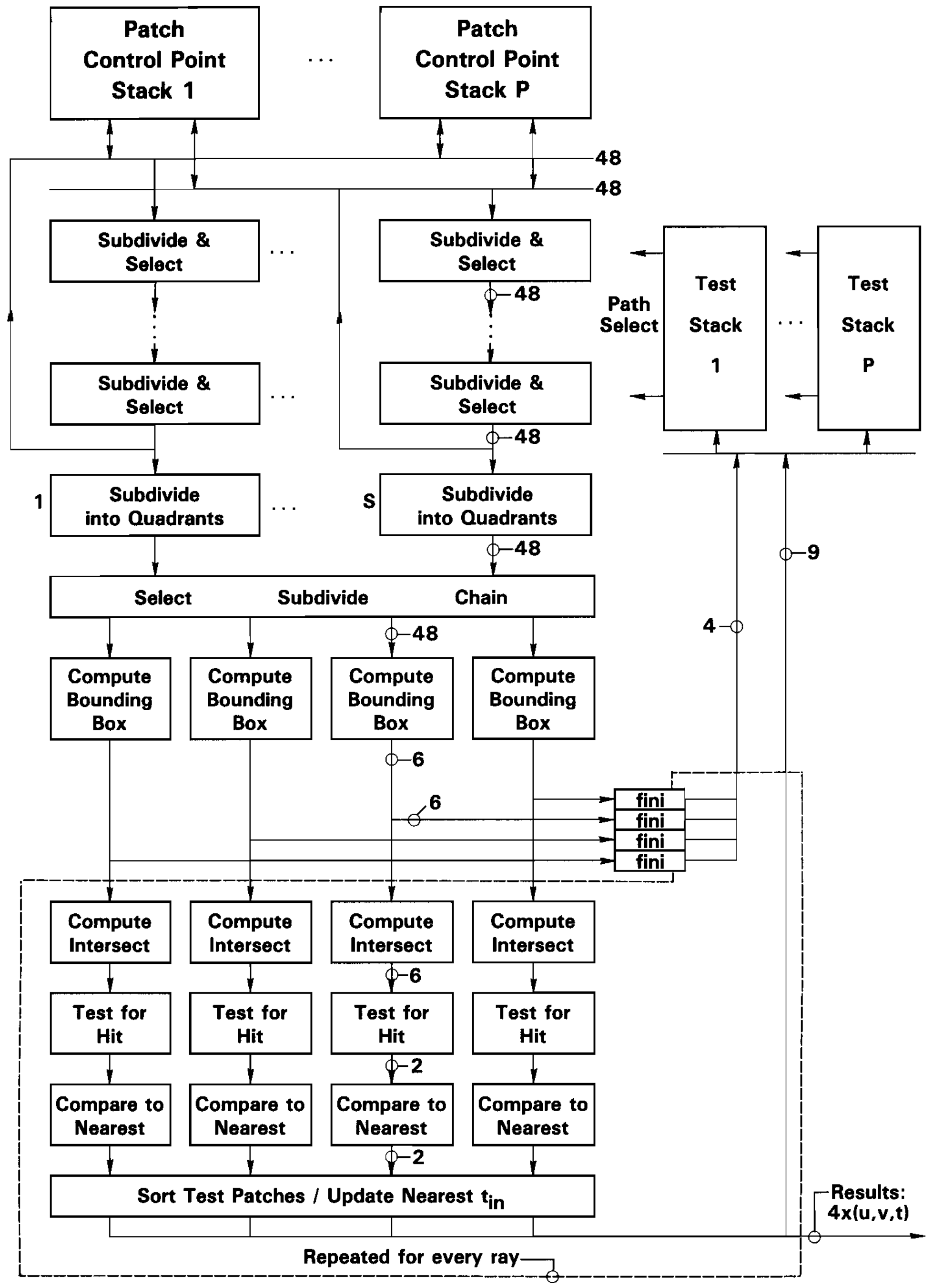
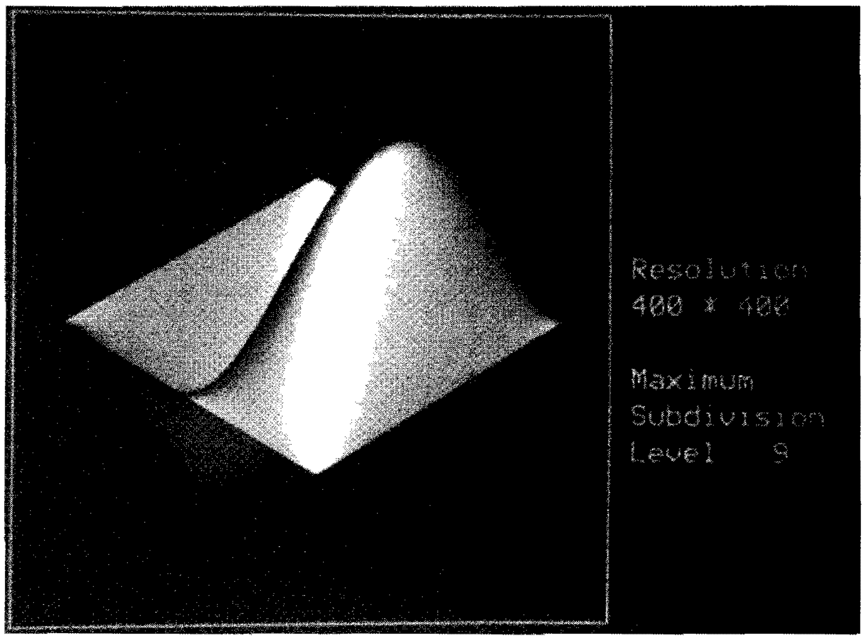


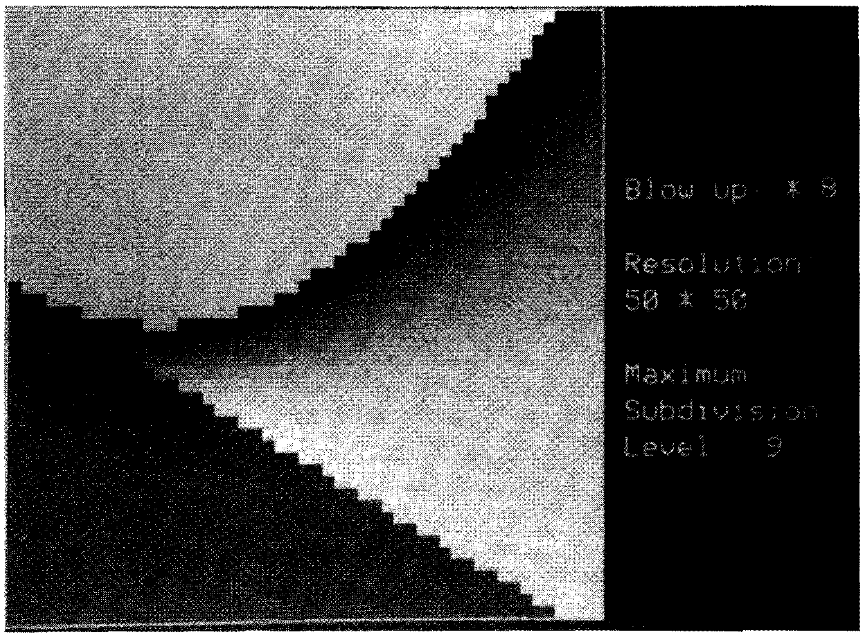
Figure 4

System Block Diagram


Resolution
400 * 400

Maximum
Subdivision
Level 9

Figure 5


Blow up * 8

Resolution
50 * 50

Maximum
Subdivision
Level 9

Figure 6

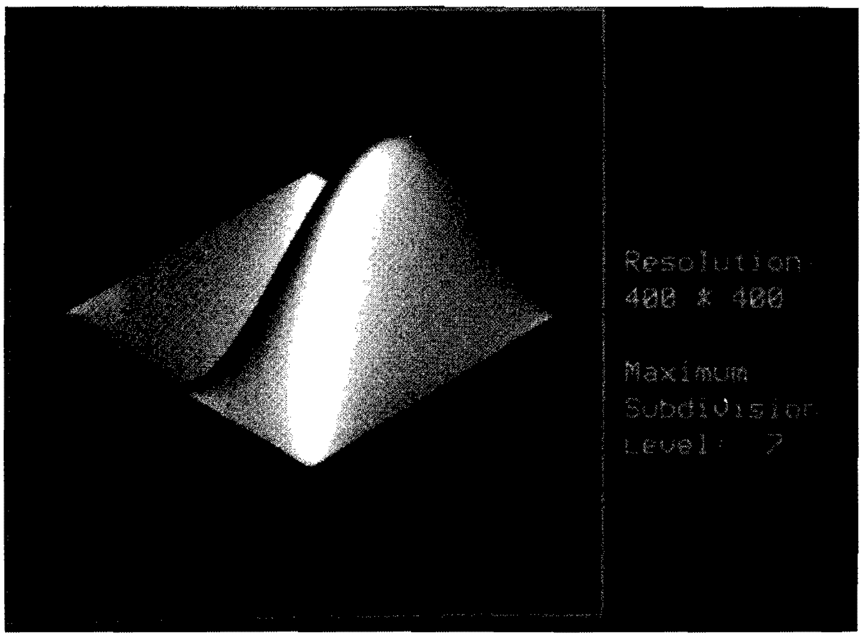


Figure 7

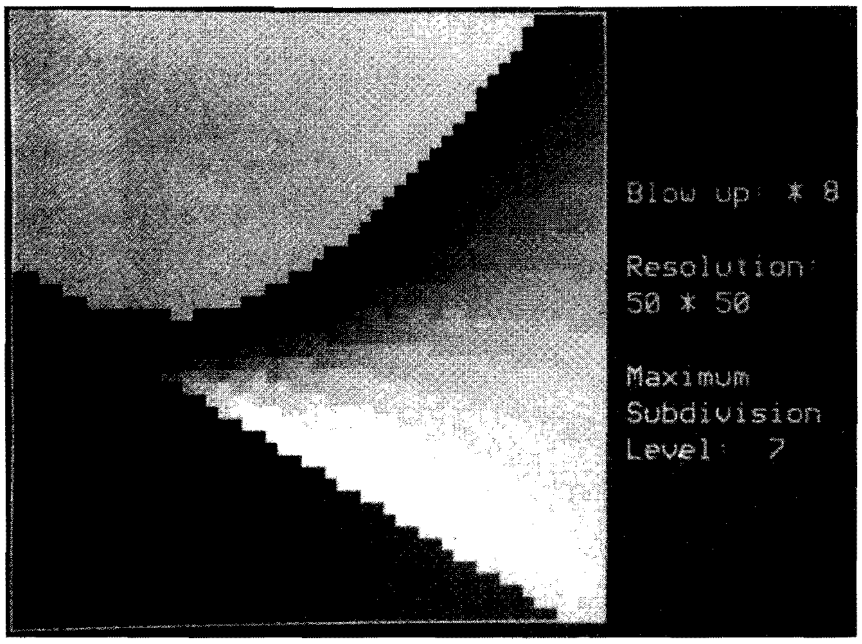


Figure 8

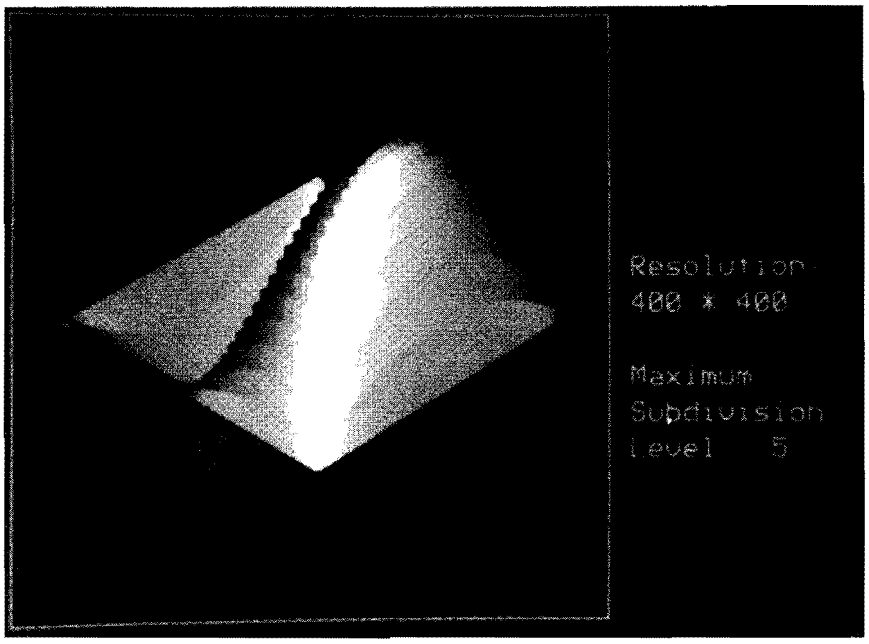


Figure 9

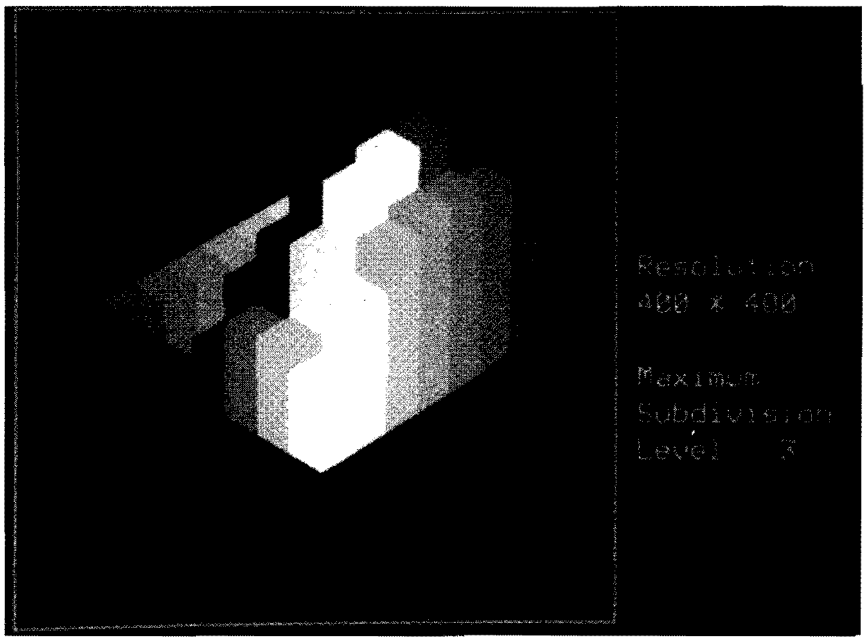


Figure 10