Hemispherical Projection for Progressive Radiosity Calculation on Massively Parallel Architectures

C. Renaud, F. Bricout, E. Leprêtre

ABSTRACT
This paper describes a massively parallel implementation of the progressive radiosity algorithm. Our algorithm is based on an hemispherical projection approach, which provides an accurate form factor approximation. As the projection plane is mapped onto a processor mesh, we propose different techniques decreasing computation time by reducing as much as possible processor inactivity. This approach successfully handles large sets of form factor sampling elements.

1.1 Introduction

During the last ten years, image synthesis has considerably developed in the fields of communication means, simulation and conception techniques. An increasing requirement of realism has appeared, which can only be obtained by taking into account global illumination phenomena. The radiosity [5] approach solves all light interactions between purely diffuse surfaces, tesselated into many planar patches. Geometric quantities, called form factors, must be computed to build an energy equation system. Solving this system provides a global viewpoint independent solution for scene illumination.

In this paper, we discuss a massively parallel approach of the progressive radiosity algorithm, based on an hemispherical projection method. This projective approach, based on the Nusselt equivalent, uses an unique projection plane and provides an accurate form factor representation. Optimization techniques have been introduced to reduce the computation time of the projection step. This reduction is especially noticeable when the number of sampling elements increases. Our approach provides an efficient use of data parallelism by successively computing all of the geometric steps onto the entire set of processing elements.

We have implemented our algorithm on a MP-1 machine to get some measures. This machine includes a scalar execution unit and a data-parallel unit. The dataparallel unit is organized as a 2D array of 1024 processing elements (PEs).

1.2 The Radiosity Method

The radiosity method [5] is based on thermal engineering theory, and requires to compute all light exchanges between every pair of objects of a scene. These computations allow to achieve the light energy balance in the scene. The move of the observer does not require any new lighting computation, as radiosity is viewpoint independent.

This method assumes that surfaces are perfectly diffuse (incident light energy is reflected with the same intensity in all directions). The surfaces are partitioned into planar patches, for which total light emission per unit area is supposed constant. At last, the energy conservation law requires a closed environment.

The radiosity equations system is :

$$B_j = E_j + \rho_j \sum_{j=0}^{N} F_{ji} B_j \quad for\ j = 1\ to\ N \tag{1.1}$$

where

. B_j is the radiosity of patch j (total light energy quantity emitted per unit area and per unit time)

. E_j its self emitted light energy (sources)

. ρ_j its reflectance factor (wavelength dependent)

. F_{ji} the form factor between the patches j and i (part of light emitted by patch j which reaches patch i, after a straight path)

. N the number of patches in the scene

Equation 1.1 shows that the light energy emitted by a patch is the sum of the patch self energy (E_j) and the part of incident energy, coming from the other patches, which is reflected by the patch. To solve this system, the form factors between each pair of patches must be computed. When it is completed, the system is solved with the Gauss-Siedel matrix inversion method [2], to get patch radiosities . This resolution diagram has two main drawbacks :

- on one hand, all form factors must be computed and stored, to allow the matrix inversion. The cost of such a storage is in $O(N^2)$, where N is the number of patches.
- on the other hand, a picture can be displayed only after form factors and radiosity computation is completed. There is not any feedback to the user before several computation hours.

Then, a reformulated radiosity algorithm has been proposed by Cohen [1], which allows to display pictures before radiosity completion. Let us note that, in the initial radiosity method, radiosity B_i of patch i is computed by gathering the energy of each patch of the environment onto the patch i. Hence, each patch radiosity must be known to be able to update patch i radiosity.

Progressive refinement radiosity reverses this process: radiosity is shot from patch i to all other patches. So, each step consists in determining patch i radiosity contribution to the radiosity of all other patches, by computing form factors F_{*i}. As energy emission decreases with time, this process converges to the same solution that the full-matrix algorithm. In order to accelerate the visual convergence, the successive shooting patches are chosen according to their energy. The most energetic patch is chosen at each step, providing the greatest lighting changes. After each shooting step, the radiosity of each patch has been updated, and it is possible to display an intermediate picture, even if the solution is not completed. Moreover, since form factors are computed on-the-fly (only one row of the form factors matrix is computing at each step), the memory cost is reduced to $O(N)$.

1.3 Form Factor Computation

Visibility between patches is the main problem in form factor computation. Several approaches, based on projective algorithms, have been proposed to compute visibility between shooting patch and all other patches in the database. This visibility is obtained by projecting the environment onto one or more surfaces, and then by applying a z-buffer operation.

1.3.1 Hemisphere Approximation

Ideally, the sampling surface is a half-sphere, but the high complexity of a depth buffer algorithm onto such a surface leads to use hemisphere approximations for projections. The hemisphere can be approximated by a hemicube [2], taking advantage of well-known planar z-buffer algorithms.

As this approach generates over-sampling and needs to compute five projections, single plane approaches have been proposed [6] [7]. The hemisphere is approximated by a very large plane parallel to the shooting patch, in order to sample most of the incident directions. As these methods also generate oversampling, different sampling patterns have been proposed to reduce the number of samples and computation time. However, those algorithms "loose" energy from grazing directions.

1.3.2 Using the Hemisphere Sustaining Disk

Form factors can be computed using the hemisphere sustaining disk in the following way. Two successive projections are performed. The first one is a central projection on the hemisphere, easily done by normalizing the coordinates of the vertices of each patch. The second one is an orthogonal projection onto the plane sustaining the hemisphere (See figure 1.1a). As the central projection of an edge onto the hemisphere is a geodesic (arc of a great circle), its orthogonal projection is an arc of an ellipse, with the same center as the hemisphere's. Using an adequate coordinates transformation (See Annexe A), the equation of an elliptic arc can be written as:

$$Au + Bv + C\sqrt{1 - u^2 - v^2} = 0 \tag{1.2}$$

As shown on figure 1.1b, the projection of an entire patch is then delimited by a set of elliptic arcs, creating a possibly concave boundary in the (u, v) space.

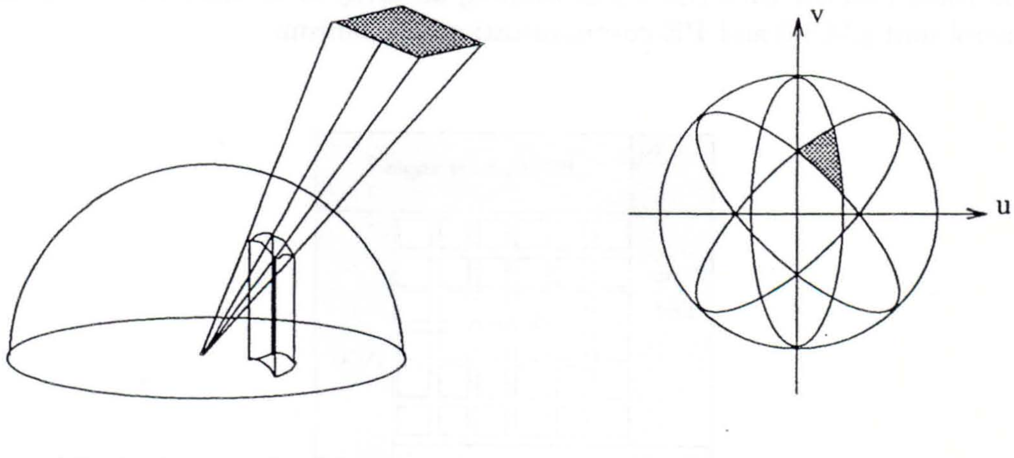

FIGURE 1.1. Use of the hemisphere sustaining disk for projection

The square enclosing the disk is divided into same sized square proxels (by analogy with *pixel*, we call *proxel* a Projected Element onto which the patch is projected). This subdivision provides the same contribution to the form factor for each proxel, and allows to use a cartesian coordinate system (u, v) for all projection operations. The problem is to determine, for each proxel, if it is inside or outside the projected patch outline. For the Pixel-Planes 5 machine, a method to determine the inside of such outlines has been introduced by Goldfeather [4]. We derive a simpler solution from his work, using sign comparisons, end providing an efficient implementation (See Annexe A for further details).

1.3.3 Advantages and drawbacks

The hemisphere approach has significant advantages compared to approximations. The unique projection plane required by this algorithm reduces the number of geometric transformations, in regard to the hemicube.

Hemispheres provides a more accurate radiosity distribution by avoiding the loss of grazing directions energy, due to single plane approximations.

Moreover, as all sampling elements contribute equally to the form factor (cf. the Nusselt equivalent), oversampling does not appear. Then, for an equal number of sampling elements, a better form factor approximation is obtained.

However, we must keep in mind that the determination of the inside of an outline made of elliptic arc is still expensive, in regard to computation time.

1.4 MP-1 Overview

The DEC MP-1 is a massively parallel single instruction, multiple data (SIMD) computer system, with one to sixteen thousand processor elements (PEs) [3]. It contains three major subsystems, which are (See figure 1.2):

. The front end processor (workstation running the ULTRIX Operating System)

. The Input/Output system

. The Data Parallel Unit (DPU) containing an array of at least 1024 PEs, an array control unit (ACU) and PE communication mechanisms.

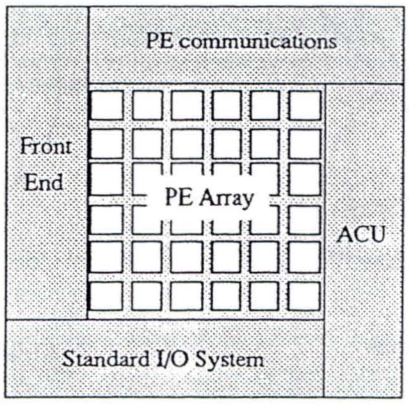

FIGURE 1.2. The MP-1 system diagram

The ACU controls the PE array. It performs operations on singular data and sends data and intructions to each PE simultaneously. Each PE is a load/store arithmetic processing element, with thirty-two 32 bits registers and 16KB of RAM, and performs both integer and floating point operations. Communications between the PEs and the ACU use a special bus. Communications between PEs use either a X-network or a Global router. The Xnet connects each PE to its 8 direct neighbours, and allows direct communications between any PEs that lies on a straight line from the original PE in one of the 8 neighbouring directions. Communications between any particular PE and a subset of other PEs use a three stage hierarchy of crossbar switches called Global Router.

1.5 A massively parallel approach

The projection of a patch onto a plane divided into several sampling elements is implicitly a massively parallel operation. However, it has to be implemented carefully in a SIMD context, in order to reach a good load-balancing and to reduce processor inactivity.

The first step of the algorithm is the elimination of patch that are trivially invisible from the source. This reduces the number of patches to take in account during the next steps. The second is geometric computations, which prepares the active patches for projection. Then the projection algorithm computes the covered proxels and performs depth buffer operations. Finally, form factors are computed and radiosity updated. An important issue of our approach is that all computation is done on the PE array. The patch database and the projecting plane are distributed over the entire set of processing elements, and the radiosity algorithm is divided into several steps, in order to eliminate idle processors. Those steps are described below.

1.5.1 Elimination of trivially invisible patch

In order to use the PE array in the best way, the visibility of every patch from the shooting patch is first computed. A patch is marked as invisible if it is entirely behind the shooting patch, or if it does not face it. Each PE computes the potentially visible patches it manages, and sorts them. As the local patch set can be seen as a stack, the sort stores the visible patches on top of the stack. This layout results in an efficient load-balancing for the following computation steps, since all the processing elements will compute an active patch.

1.5.2 Projection parameters computation

As projection parameters computation is the same for each patch, it is applied simultaneously onto the full PE array, taking advantage of the SIMD architecture. Geometric transformations are first applied on the patches. Their coordinates are transformed into the source patch coordinate system, and clipping is performed using the $z = 0$ plane. Projection parameters (elliptic arc equations, initial values, ...) are then computed and stored with the patch data. After this process, N patches are ready for projection, where N is the number of available processors in the array.

1.5.3 The projection step

As this is the most computationally demanding stage of the algorithm, care must be taken in order to ensure an efficient load-balancing over the processors. To determine which proxels are covered with the projected patch, we have to compute, for each one, whether it is inside or outside the outline defined by elliptic arcs. Each arc equation is evaluated for each proxel, setting an activity flag to false if the proxel is on the wrong side of the arc. After all evaluations are done, a proxel is inside the projected outline if its activity flag is still set to true. Note that as the square root that appears in equation 1.2 only depends on the proxel coordinates, it is computed once when initializing the projection plane, and then stored as part of a proxel data. A proxel will also contain a patch identity, The depth buffer operation is performed for each inner proxel (u, v), by computing the true distance between the proxel and the patch, and comparing it with the previously stored value. A detailed explanation about the distance computation is given in Appendix A.

This projection process is applied one patch at a time, using all available computation resources. This provides a more efficient use of the SIMD approach than by simultaneously projecting different patches, since the computation volume is different from one patch to another. As each PE manages a different patch, each processor has to diffuse sequentialy its local parameters to all the other processors. When receiving a patch, the processors compute its projection and wait for the next patch parameters. Diffusion is performed using the communications between PEs and ACU, because the Xnet and the Global Router are not well suited for diffusion. To diffuse its patch parameters, a PE sends them to the ACU, which diffuses them to all the PE array.

1.5.4 Form factor computation

The last stage of the algorithm evaluates form factor and updates radiosity. In order to compute a patch form factor, the number of proxels the patch covers must be known, but proxels are distributed over the PE array. So, a patch form factor must be propagated toward each PE, in order to be sure that each proxel covered by the patch has been taken in account. Propagation is performed using local communications between PEs (*xnet* instruction). A PE receiving a patch form factor searches this patch in its local proxels, and increases the form factor when the patch covers some of them. This operation is done for N patches at a time.

When the form factors have been computed, each PE updates the radiosity of its local patch, and a *reduce* instruction is applied on those values in order to determine the most energetic patch in the current set.

Those operations are performed for each level of patch stack, the final most energetic patch being chosen among the most energetic patches selected into the previous levels. This patch will be used in the next radiosity shooting step, according to the progressive radiosity algorithm.

The following pseudocode procedure resumes the organisation of the algorithm.

```
While not convergence do
    compute and sort the visible patches
    For each visible patch stack level do
        compute the projection parameters
        For each current patch /* one per PE */
            diffuse the local projection parameters
            apply the projection process
        EndFor
    EndFor
    compute form factors and update radiosities
    choose the new shooting patch
EndWhile
```

1.5.5 Proxel distribution and bounding boxes

As we do not have as much PEs as proxels, each processor has to manage several proxels. Since all processors have to project the same patch at a time, contiguous proxels are distributed on neighbour processors (using a cyclic distribution). The projection plane is cut into projection areas, each one containing as many proxels as PEs. This approach limits the number of idle processors, compared to contiguous proxel zone approach (see figure 1.3).

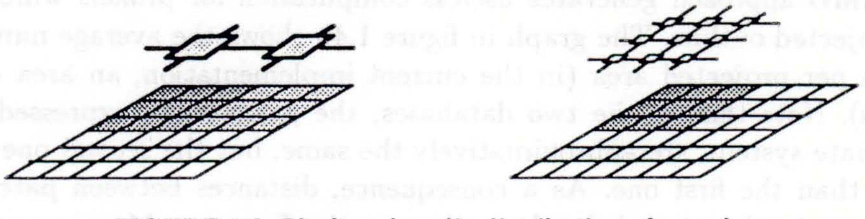

FIGURE 1.3. Block and cyclic distribution of proxels

The projection equations have to be evaluated for each proxel. However, the average area covered by patch projections is generally small in regard to the projection plane. So, we evaluate the projection equations only in a rectangular area, bounding the projection outline. This removes all projection zones where the patch is not projected. When the number of areas increases with precision requirements, this method greatly accelerates the projection process. As the projected outline is elliptic arc based, we propose, in Appendix B, a method to compute the bounding rectangle of such outlines.

1.6 Results

Our algorithm has been applied on different databases. Each one of them represents various illumination conditions and numbers of patches. We present here some results for two different databases, containing 2,000 and 14,000 patches. For each one, we measured the average computation time per shooting stage and the number of proxels inside the projected outline per area (See figure 1.4). Resolution of the projection plane has regularly been increased, in order to improve sampling precision (a resolution value N means that the square enclosing the projection circle is divided into $N \times N$ square proxels).

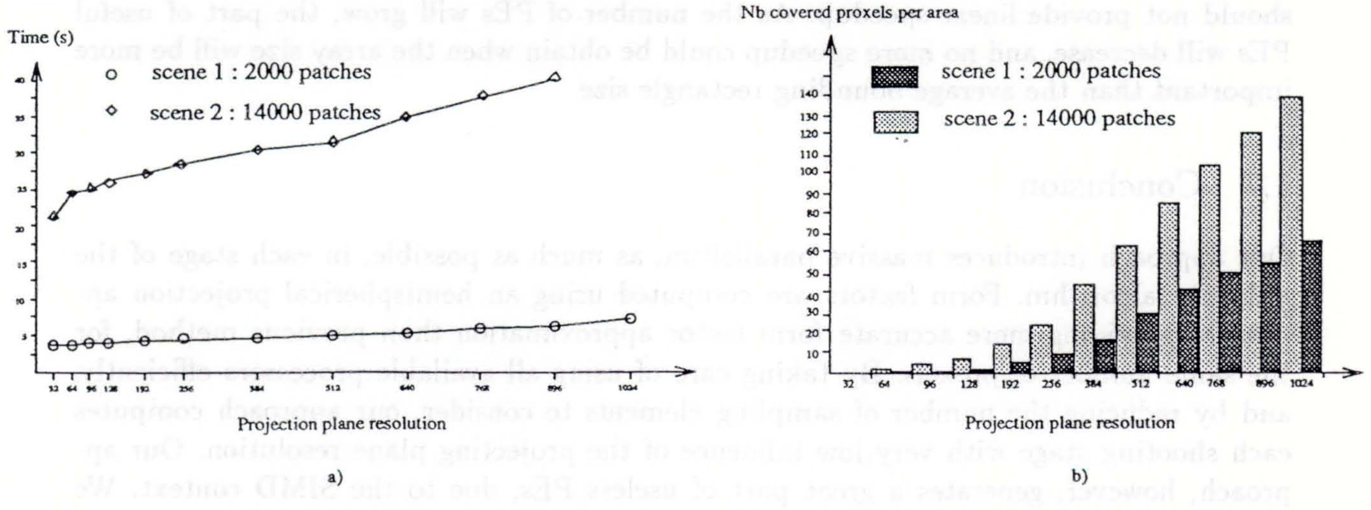

FIGURE 1.4. Computation times (a) and average number of useful PEs (b)

Results first show that, as the number of sample elements increases, computation time grows particularly slowly, even for very high resolutions. This is a direct effect of the bounding rectangles, which limits dramatically the number of areas to consider for projection. For example, a 1024×1024 projection plane is split into 1024 (32×32) areas, and only 3 to 5 of them are concerned with each patch projection.

However, computation remains time consuming. This can be explain by three factors:

- The SIMD approach generates useless computation for proxels which are outside the projected outline. The graph in figure 1.4b shows the average number of useful proxels per projected area (in the current implementation, an area covers 32×32 proxels). Note that in the two databases, the patch sizes, expressed in the same coordinate system, are approximatively the same, but the second one is nine times bigger than the first one. As a consequence, distances between patches are more important in the second database than in the first one. Moreover, the projected surfaces are smaller, and there are less useful PEs.

- The patch diffusion, from one processor to all others, takes a great part of the total computation time, even when using the fastest diffusion scheme. The patch projection parameters (including patch sustaining plane, elliptic arcs equations, patch identity, bounding rectangles, initial and incremental values) contains about 100 bytes of data. The diffusion time for 1024 patch parameters is approximatively of 0.75 second, and can not be reduced. Let us note that the total diffusion time only depends on the total number of patches in the database. This time will not change when the PE array size will grow.

- Elliptical and distance evaluations require floating point operations that are time consuming on the MP-1, because of the 4 bits PE architecture. Assuming triangular patches, each proxel computation needs 9 products and 6 summations for the inside evaluation, and 3 products and 2 summations for the distance computation. Nevertheless when a projected patch overlaps several areas, those operations are performed only for the first area, the next ones being computed using summations of delta values.

Those results have been obtained on a 32 × 32 PEs machine, and are to be tested on a 128 × 128 PEs configuration. However, using a more powerful MP-1 configuration should not provide linear speedup. As the number of PEs will grow, the part of useful PEs will decrease, and no more speedup could be obtain when the array size will be more important than the average bounding rectangle size.

1.7 Conclusion

Our approach introduces massive parallelism, as much as possible, in each stage of the radiosity algorithm. Form factors are computed using an hemispherical projection approach, providing more accurate form factor approximation than previous method, for the same number of proxels. By taking care of using all available processors efficiently, and by reducing the number of sampling elements to consider, our approach computes each shooting stage with very low influence of the projecting plane resolution. Our approach, however, generates a great part of useless PEs, due to the SIMD context. We have so to study more precisely the different steps of the algorithm, in order to balance computations and communications tasks and to obtain a greatest efficiency.

1.8 Appendix A: Derivations for hemispherical projection

Goldfeather proposes in [4] to transform patch edges defined in the (x, y, z) space into a (u, v, ρ) space defined as follows:

$$\left(\begin{array}{l} \rho = \sqrt{x^2 + y^2 + z^2} \\ u = \frac{x}{\rho} \\ v = \frac{y}{\rho} \end{array}\right. \Longleftrightarrow \left(\begin{array}{l} x = \rho u \\ y = \rho v \\ z = \rho\sqrt{1 - u^2 - v^2} \end{array}\right.$$

Projecting an edge onto the hemisphere sustaining plane is the same as to compute a geodesic. This geodesic is the intersection between the hemisphere and the plane defined by the origine of the sphere and the two vertices of the edge. The plane equation is as follows:

$$\Pi : Ax + By + Cz = 0$$

By replacing (x, y, z) with (u, v, ρ), we get the equation of the projected ellipse:

$$E(u, v) : Au + Bv + C\sqrt{(1 - u^2 - v^2)} = 0$$

However, this equation does not describe the entire ellipse, but only a half one. This can be easy understood by remembering that an ellipse has a quadratic form. The entire ellipse equation is found by squaring equation 1.2.

This equation describes the half of the ellipse where we can found the projected edge. We will call this half the Positive Half Ellipse (PHE), as the edge that projects on it is above the $z = 0$ plane.

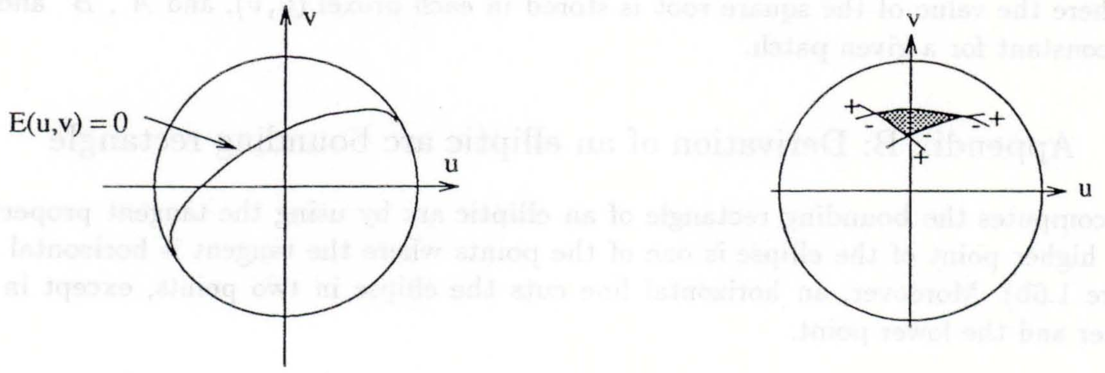

FIGURE 1.5. Use of the PHE a) a PHE b) the inside sign

The PHE cut out the projection circle into two parts, the sign of $E(u, v)$ being different in each part (see figure reffig:phe). One of those regions contains to proxels that are inside the projected patch, in regard to the plane Π equation, applied to the current edge.

As each edge projection defines such circle division, a proxel $(u0, v0)$ is inside the projected patch if it is inside each PHE, that is if $E(u0, v0)$ has the same sign as an inside proxel for each edge. However, it is not necessary to know an inner point to compute all the inside proxels. We assume that all the patches have the same orientation, clockwise or counterclockwise. Then evaluation of the $E(u, v)$ equations for each edge gives the same sign for the inside of the outline (positive for clockwise orientation and negative for counterclockwise orientation).

For each proxel $(u0, v0)$, we have then to compute a value $E(u0, v0)$ for each projected egde, and to compare the sign of these values to the predefined inside sign. If all the signs are the same that the inside's, the depth buffer comparison is performed.

Spencer [8] suggests to use a constant value of the distance between the hemisphere center and each point of a patch. However, this is only correct for very small or very distant patches. The true distance is also computed for each point of a patch [4]. Taking a point P of coordinates (x, y, z), its distance to the hemisphere center is $\rho = \sqrt{x^2 + y^2 + z^2}$. Let us note

$$\Pi : A'x + B'y + C'z + D' = 0$$

the patch sustaining plane equation. Using the previous coordinates transformation, this equation is rewritten:

$$\Pi' : A'\rho u + B'\rho v + C'\rho\sqrt{1 - u^2 - v^2} + D' = 0$$

where u and v are the point P projected coordinates (coordinates of the proxel where the distance has to be computed). Distance from this proxel to the point P is given by

$$\rho = -\frac{D'}{A'u + B'v + C'\sqrt{1 - u^2 - v^2}}$$

As only a relative distance is needed, assuming that it grows when ρ grows, the distance is computed using the following expression:

$$\rho' = -\frac{1}{\rho} = A''u + B''v + C''\sqrt{1 - u^2 - v^2}$$

where the value of the square root is stored in each proxel (u, v), and A'', B'' and C'' are constant for a given patch.

1.9 Appendix B: Derivation of an elliptic arc bounding rectangle

We computes the bounding rectangle of an elliptic arc by using the tangent properties. The higher point of the ellipse is one of the points where the tangent is horizontal (see figure 1.6b). Moreover, an horizontal line cuts the ellipse in two points, except in the higher and the lower point.

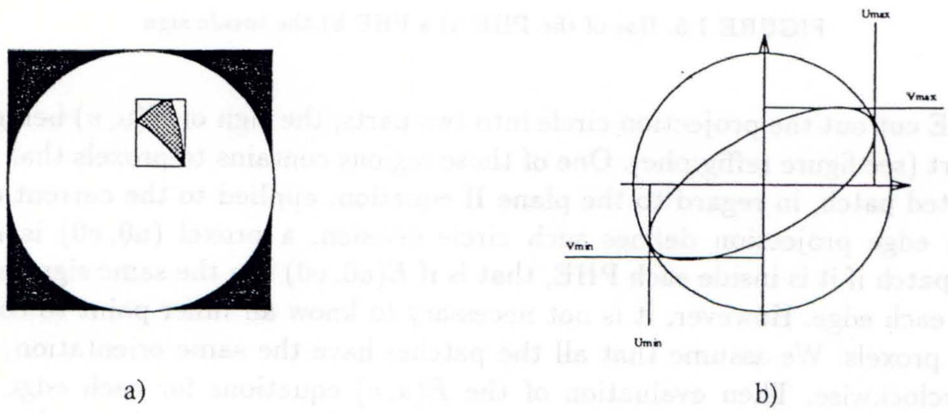

FIGURE 1.6. Bounding rectangle (a) and extrema values for an ellipse (b)

By computing the intersection between the ellipse and a line V, and by reducing the solution set to a single intersection, we can find the two possible values V_{min} and V_{max}:

$$V_{max} = \sqrt{\frac{A^2+C^2}{A^2+B^2+C^2}} \quad V_{min} = -\sqrt{\frac{A^2+C^2}{A^2+B^2+C^2}}$$

where A, B and C are the coefficients of the plane defined with the two vertices of the edge and the center of the hemisphere.

By replacing those values in the $E(u, v)$ equation, we find the corresponding values of u, and then the higher and lower points:

$$P_{max} = \left(\frac{-ABV_{max}}{A^2+C^2} \, , \; V_{max} \right) \quad P_{min} = \left(\frac{-ABV_{min}}{A^2+C^2} \, , \; V_{min} \right)$$

Using the same way, we can derive the points corresponding to the most left and the most right points of the ellipse, using vertical tangents. The values we obtain are then:

$$U_{max} = \sqrt{\frac{B^2+C^2}{A^2+B^2+C^2}} \quad U_{min} = -\sqrt{\frac{B^2+C^2}{A^2+B^2+C^2}}$$

$$P_{right} = \left(\frac{-ABU_{max}}{B^2+C^2} \, , \; U_{max} \right) \quad P_{left} = \left(\frac{-ABU_{min}}{B^2+C^2} \, , \; U_{min} \right)$$

As this bounding rectangle bounds the entire ellipse (See figure 1.6a), simple tests are then required to get the bounding rectangle of the elliptic arc. We explains here the method for the higher value of V. It can be generalized for the other values we search.

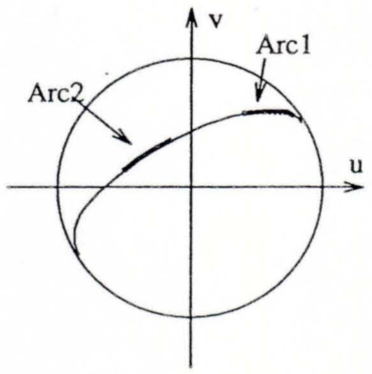

FIGURE 1.7. The two possible cases for the elliptic arcs

Two cases are possible, depending on the position of the projected arc onto the ellipse. In the first case (arc 1 of figure1.7), an extrema exists between the two extremities of the arc. Then, it must be used as the higher value of the bounding rectangle, the lower value being one of the two extremities V value. In the second case (arc 2), no extrema exists between the two points. The higher v value of the two points is choosen, the lower value giving the lower value of the bouding rectangle.

The global bounding rectangle of a projected patch is the union of all the edge bounding rectangles.

1.10 References

[1] M.F. Cohen, S.E. Chen, J.R. Wallace, and D.P. Greenberg. A progressive refinement approach to fast radiosity image generationthesis. *SIGGRAPH'88*, 22(4):75–84, August 1988.

[2] M.F. Cohen and D.P. Greenberg. The hemicube: a radiosity solution for complex environments. *SIGGRAPH'85*, 19(3):31–40, July 1985.

[3] Digital Equipment Corporation. *DECmpp Architecture Specification*, January 1992.

[4] J. Goldfeather. Progressive radiosity using hemispheres. Technical Report TR 89-002, University of North Carolina at Chapel Hill, February 1989.

[5] C.M. Goral, K.E. Torrance, D.P. Greenberg, and B. Battaile. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH'84*, 18(3):213–22, July 1984.

[6] R.J. Recker, D.W. George, and D.P. Greenberg. Acceleration techniques for progressive refinement radiosity. *Computer Graphics*, 24(2):59–66, March 1990.

[7] F. Sillion and C. Puech. A general two-pass method integrating specular and diffuse reflection. *SIGGRAPH'89*, 23(3):335–344, August 1989.

[8] S.N. Spencer. The hemisphere radiosity method: a tale of two algorithms. *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 127–135, June 1990.