THE MAGIC OF THE Z-BUFFER: A SURVEY

Theoharis Theoharis, Georgios Papaioannou, Evaggelia-Aggeliki Karabassi

Department of Informatics, University of Athens,
Panepistimioupolis, Ilisia, Athens, 15784
Greece
theotheo@di.uoa.gr http://graphics.di.uoa.gr/~graphics

ABSTRACT

The wide availability of hardwired Z-buffer has sparked an explosion in the number of applications of the algorithm whose origins lie in hidden surface elimination. This paper presents a much-needed survey of key applications of the Z-buffer from the fields of rendering, modelling and vision in a common notation in order to help users make better use of this resource.

Keywords: Z-buffer, depth-buffer, rendering.

1. INTRODUCTION

When the depth buffer (Z-buffer) algorithm was originally proposed in the middle 1970's as a hidden surface elimination technique [Catmu74], most people dismissed it as impractical due to its "huge" memory demands. Today the Z-buffer is a standard feature in all graphics cards, due mainly to its generality and simplicity, which makes it easily implementable in VLSI. The wide availability of the hardwired Z-buffer and its embodiment in well known graphics libraries such as OpenGL [Segal99] has pushed its application domain beyond the realm of hidden surface elimination. It must be noted that the Z-buffer is one of the few pieces of hardwired logic that is so commonly available. Several novel and ingenious methods have been proposed ranging from symmetry detection to voxelisation. Many of them originate from our Computer Graphics Laboratory [CGL00].

In this paper, we present the applications of the Z-buffer that we consider most interesting, giving references where necessary for further details. We do not discuss its well-known use for hidden surface elimination. To our knowledge this is the first survey on this important and very practical subject. We first define some common notation that will be used in the description of the applications.

Assume a right handed *world coordinate system* centered at $\overline{O} = (0,0,0)$ and the unit vectors $\overline{X}, \overline{Y}, \overline{Z}$ along the 3 major axes. The operation of the Z-buffer is captured by a function Z which, given a set of parameters, returns a mapping from 2D pixel coordinates to depth value:

$$zmap : num \times num \rightarrow num$$

$$zmap = Z(\overline{C}, \vec{V}, \vec{U}, \vec{N}, near, far, operator)$$

where $\overline{C}, \vec{V}, \vec{U}, \vec{N}$ specify a *viewing co-ordinate system* centred at the point where an imaginary observer is, *near* and *far* specify the clipping limits (distances from $\overline{C}$) which allow normalisation of the depth values and *operator* is the Z-buffer test which, following OpenGL, can be one of $\{LESS, EQUAL, GREATER, GEQUAL, LEQUAL\}$.

For example:

$$zexample = Z((1,2,6), \overline{X}, \vec{Y}, \vec{Z}, 5, 10, LESS)$$

specifies a Z-buffer calculated from $\overline{C} = (1,2,6)$ with viewing axes parallel to the respective world co-ordinate axes, minimum and maximum allowed Z depths of 6+5 and 6+10 respectively (allowing for the normalisation $(11 \rightarrow 0, 16 \rightarrow 1)$) using the *LESS* test (strictly smaller values pass).

The applications presented below cover rendering (paragraphs 2,3,4), modelling (paragraphs 5,6) and vision (paragraphs 7,8).

2. IMAGE COMPOSITING

It is often necessary to combine objects rendered separately into a single image. One might, for example, prefer to use a certain package for human animation and a different one for producing smoke and clouds out of particle systems. If depth and viewpoint conditions are respected, the various objects may be rendered in separate pairs of frame and Z-buffers and then combined into the final image, eliminating hidden surfaces [Porte84, Duff85]. Suppose we have *N* separately rendered objects whose Z-buffers are calculated with the same viewing characteristics:

$$z_1 = Z(\bar{C}, \vec{V}, \vec{U}, \vec{N}, near, far, LESS)$$
$$z_2 = Z(\bar{C}, \vec{V}, \vec{U}, \vec{N}, near, far, LESS)$$
...
$$z_N = Z(\bar{C}, \vec{V}, \vec{U}, \vec{N}, near, far, LESS)$$

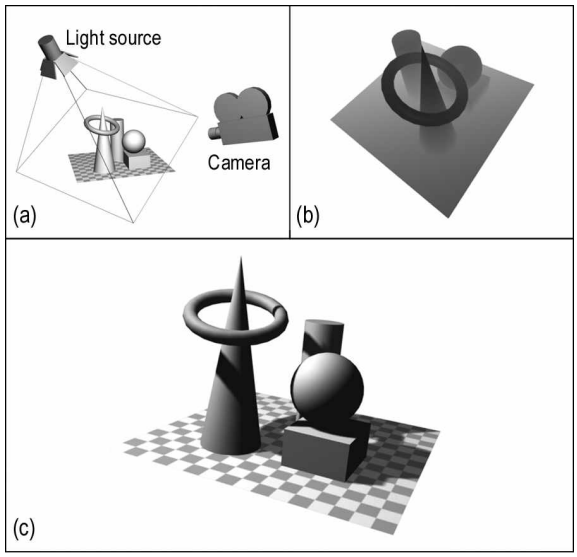
If the respective frame buffers are $f_1 \ldots f_N$, then the final depth-merged frame and Z-buffers f_0 and z_0 can be computed with the following piece of pseudocode:

```
for each pixel (x,y)
    z0(x,y) = z1(x,y)
    f0(x,y) = f1(x,y)
    for i = 2(1)N
        if zi(x,y) < z0(x,y)
            z0(x,y) = zi(x,y)
            f0(x,y) = fi(x,y)
```

A similar idea is used to place a 3D cursor in a 3D scene, hiding parts of it that are occluded by scene objects [Foley91]. The frame and Z buffers of the cursor are simply combined with those of the scene in the above manner, except that in this case the contents of the scene Z buffer are not altered. Also the operation need not span the whole image area but only the cursor's bounding box.

3. SHADOW MAPS

The most popular alternative use of the Z-buffer algorithm is the fast generation of shadows from spotlights or parallel projectors. This shadow generation technique was first introduced by Williams in 1978 [Willi78] and its variations are still widely used in rendering software and real-time applications, like computer games.



Generating shadows with the Z-buffer
Figure 1

Let a spotlight or projector with a lighting range $range_L$ be placed at point $\bar{L}$ in the scene and point along the direction $\vec{N}_L$ (Fig. 1a). Let also $\mathbf{M}_L$ and $\mathbf{P}_L$ be the geometric transformation and projection matrices of the light source ($\bar{L}, \vec{V}_L, \vec{U}_L, \vec{N}_L$). The shadow test itself is very simple and is divided into the following two steps:

- Render the scene from the light-source's point of view $\bar{L}$ using $\vec{N}_L$ as camera axis (Fig. 1b) and store the corresponding depth-map (*shadow-map*):
  $Z_L = Z(\bar{L}, \vec{V}_L, \vec{U}_L, \vec{N}_L, 0, range_L, LESS)$

- Revert to the normal camera view and render the scene (Fig. 1c). A point $\bar{P}$ on a surface is shadowed if it is located at greater distance than the value stored in Z_L, when $\bar{P}$ is expressed in the light-source's viewport coordinates: $\bar{P}' = \mathbf{P}_L \cdot \mathbf{M}_L^{-1} \cdot \bar{P}$, that is, if $z' > Z_L(x', y')$.

If z' is outside the range $[0, range_L]$, $\bar{P}$ is considered to be in shadow.

In order to apply this method to non-directional lights, multiple shadow-maps must be combined in a spherical or cubic map so that any direction on the unit sphere centred at $\bar{L}$ is addressable. If multiple light sources participate in the scene, the above procedure is duplicated for each of them.

The main advantages of Z-buffer-based shadows are the simplicity and generality of the

algorithm, its speed as well as the ability to produce soft shadows. Today's graphics hardware can take advantage of single-cycle multi-pass rendering to provide good quality shadows in real-time, even with a high polygon count.

The most noticeable problem of the shadow-maps is the production of aliasing effects. The regular sampling of the Z-buffer may produce annoying jagged shadow edges, especially in low buffer resolutions. Because of the discrete form of the shadow-map, $Z_L(x', y')$ must be interpolated by filtering the map over an area of map cells in the neighbourhood of (x', y'). This interpolation, if not performed carefully, can lead to bad distance comparisons and therefore, misplaced shadows. Solutions to the above aliasing problems have been proposed, as in [Cook86] and [Reeve87].

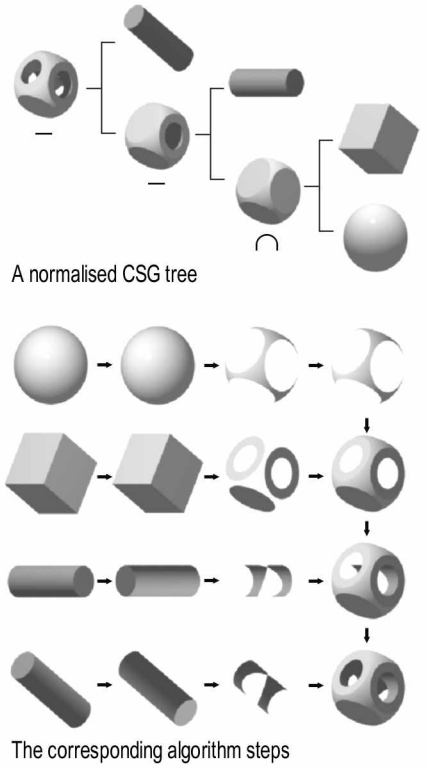
A generic drawback of Z-buffer shadow generation is the inability of the (original) algorithm to handle filtered-shadows from partially transparent surfaces or volume data, a problem, which other methods like ray-traced shadows successfully tackle. However, due to its speed and ability to effectively produce soft shadows, it is the most commonly used shadow generation technique in commercial programs.

4. CSG RENDERING

Constructive Solid Geometry (CSG) is a very common method of modelling complex 3D objects from simple primitives by performing Boolean operations on their volumes. The relationship between the primitives is described as a *CSG tree* whose leafs are the primitive objects and intermediate nodes represent the partial Boolean operations. The root of such a tree represents the final CSG object.

Rendering of the CSG object can be done after the surface boundary of the final object is evaluated by displaying the resulting surface elements (e.g. clipped and tessellated triangles) or before the actual surface generation, in *image-space*. The later category includes ray-casting, scan-line and Z-buffer-based techniques.

Z-buffer methods utilise the standard Z-buffer algorithm implemented in conventional graphics hardware in order to render a CSG hierarchy using multiple passes of clipping (stencil test) and depth sorting (depth test) operations. Such algorithms are proposed in [Goldf86, Epste89, Wiega96, Stewa98, Stewa00].

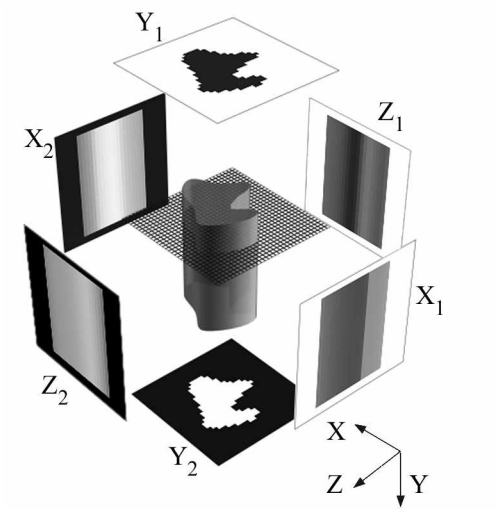


The CSG algorithm by Goldfeather et al
Figure 2

A key point to the Z-buffer CSG rendering techniques is the conversion of the initial arbitrary CSG tree into a *normalised CSG tree*. A CSG tree is normalised when it is transformed in a sum-of-products form (union of primitive differences or intersections). A product may consist of a primitive and a normalised CSG sub-tree.

A representative algorithm of the Z-buffer-based CSG rendering category is the one proposed by Goldfeather et al [Goldf86] and implemented in non-specialised hardware by Wiegand [Wiega96]. In this algorithm, after bringing the CSG hierarchy in the form of a normalised CSG tree, each (convex) primitive surface is clipped in image-space by its siblings using the depth and stencil tests. The partial surfaces are finally merged with a simple Z-buffer operation (z-less-than) (Fig. 2).

The technique uses one Z-buffer (*surface Z-buffer*) to store the visible surface of each primitive and another one (*output Z-buffer*) to compose in correct depth-order the partial results of the surface Z-buffers. A stencil buffer is also necessary for the clipping of the primitives in the *surface Z-buffers*. Each time a primitive is compared with the *surface Z-buffer*, the stencil test is configured so that the stencil buffer holds the number of surfaces in front of the Z-buffer already stored. The general algorithm is as follows:



Buffer setup for voxelisation.
Figure 3

Clear the *output Z-buffer* to *z=far*.
For each CSG tree product *P*:
  For each primitive *A* in *P*:
    Clear the *surface Z-buffer* to *z=far*.
    If A is subtracted
      Draw back of *A* into *surface Z-buffer*.
    Else
      Draw front of *A* into *surface Z-buffer*.
    For all other primitives *B* in *P*:
      Update the stencil buffer and mask the *surface Z-buffer* appropriately:
      If *B* is subtracted
        Accept pixels with even stencil value.
      Else
        Accept pixels with odd stencil value.
    Draw the *surface Z-buffer* into the *output Z-buffer* where *surface z* value < current *output z* value.

Stewart and Leach [Stewa98] introduced the idea of grouping the primitives into layers in order to perform more than one writes to the surface Z-buffer in a single pass. The generation of such layers depends on the *depth complexity* of the composite object under a certain angle of observation. The term depth complexity refers to the number of overlapping surfaces in the z-direction.

Z-buffer CSG rendering allows the interactive previewing of Boolean operations on 3D objects before the (often expensive) computation of the actual CSG surface geometry is performed. In case of non-convex primitives, object-space methods can be used to partition the primitives into convex ones.

5. VOXELISATION

Volume graphics is a domain that has a rather short history, but is gaining increasing popularity and voxel-based models are currently used in a variety of applications. Voxelisation is the process of approximating a continuous object by a set of voxels, which consists of sampling the initial object and assigning a value to each voxel of a three dimensional raster.

The z-buffer has been used in a simple and very fast binary voxelisation algorithm, which rapidly produces volume data from any type of original model [Karab99].

The algorithm assumes that the object to be voxelised is surrounded by a bounding box, and that each face of the box is a viewing plane. A depth buffer is generated for each face by a parallel projection of the object onto it (see Fig. 3). Assuming the bounding box is centred at the axes origin and its dimensions are d_X, d_Y and d_Z along the three major axes, the following six depth buffers are generated:

$$X_1 = Z\left(\bar{C}_X, -\vec{Z}, -\vec{Y}, \vec{X}, 0, d_X, LESS\right)$$

$$X_2 = Z\left(\bar{C}_X, -\vec{Z}, -\vec{Y}, \vec{X}, 0, d_X, GREATER\right)$$

$$Y_1 = Z\left(\bar{C}_Y, \vec{X}, \vec{Z}, \vec{Y}, 0, d_Y, LESS\right)$$

$$Y_2 = Z\left(\bar{C}_Y, \vec{X}, \vec{Z}, \vec{Y}, 0, d_Y, GREATER\right)$$

$$Z_1 = Z\left(\bar{C}_Z, \vec{X}, -\vec{Y}, \vec{Z}, 0, d_Z, LESS\right)$$

$$Z_2 = Z\left(\bar{C}_Z, \vec{X}, -\vec{Y}, \vec{Z}, 0, d_Z, GREATER\right)$$

where $\bar{C}_X = (-d_X/2, 0, 0)$, $\bar{C}_Y = (0, -d_Y/2, 0)$ and $\bar{C}_Z = (0, 0, -d_Z/2)$

For each pair of bounding box faces the algorithm obtains two values for each pixel from the two buffers, representing the minimum and maximum distance between the object and the view plane. As a result, any given voxel centre will have three pairs of values associated with it, one pair per axis. If a voxel's location is bounded by all three pairs, then the voxel is inside the object.

The method presents the limitation that it can miss concavities: if some area of the surface is not visible from any of the six faces, then this area will not be properly voxelised. While this is a disadvantage over more accurate (and slow) methods, in practice the majority of objects can be quickly and successfully voxelised by this algorithm.

The algorithm described above voxelises the entire object. A variation exists to voxelise only the surface of the object.

6. DISCRETE VORONOI DIAGRAMS

Hoff et al [Hoff99] propose a method for the computation of discrete Voronoi diagrams, i.e. Voronoi diagrams over a discrete space such as the image plane, using the Z-buffer.

Let the discrete space be the image plane and suppose that the Voronoi sites [Voron08] are points distributed over the image plane. One way to compute the Voronoi diagram is:

for every pixel *p*
$d_{\min} = MAXINT$
for every Voronoi point *v*
$d = dist(v, p)$
if $d < d_{\min}$
$d_{\min} = d$
$colour(p) = colour(v)$

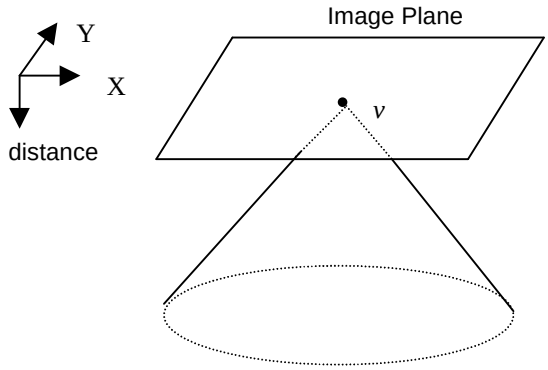
The above algorithm colours every pixel with the colour of the nearest Voronoi site. Alternatively the algorithm may be rearranged to iterate through the sites, computing distances to all pixels, and updating the colour and minimum distance (kept per pixel) accordingly. Now this sounds very much like the Z-buffer and indeed it is. All that is necessary is to create a distance function $dist(v,(x,y))$ which yields the distance of pixel (x,y) from site *v*. For point sites this is simply a right circular cone with its apex at site *v*, see Fig. 4.

The cone can be approximated by a set of triangles proceeding radially outward from the apex. Hoff et al estimate that in order to achieve subpixel accuracy in a 512x512 resolution image plane we only need 60 triangles.

These approximated cones are rendered, each in different colour, using the Z-buffer with parallel projection and the result is the 2D discrete Voronoi diagram of the point sites. Hoff et al generalise the method to line-segment- and polygon-sites as well as to 3D.

7. OBJECT RECONSTRUCTION

The Z-buffer algorithm, apart from its many uses in the computer graphics domain, has been also applied to solve computer vision problems. Such a case is presented in [Papai00a], where the reconstruction of objects from their fragments is attempted. In this procedure, object fragments have to be tested one against another for complementary matching in order to be glued together later.

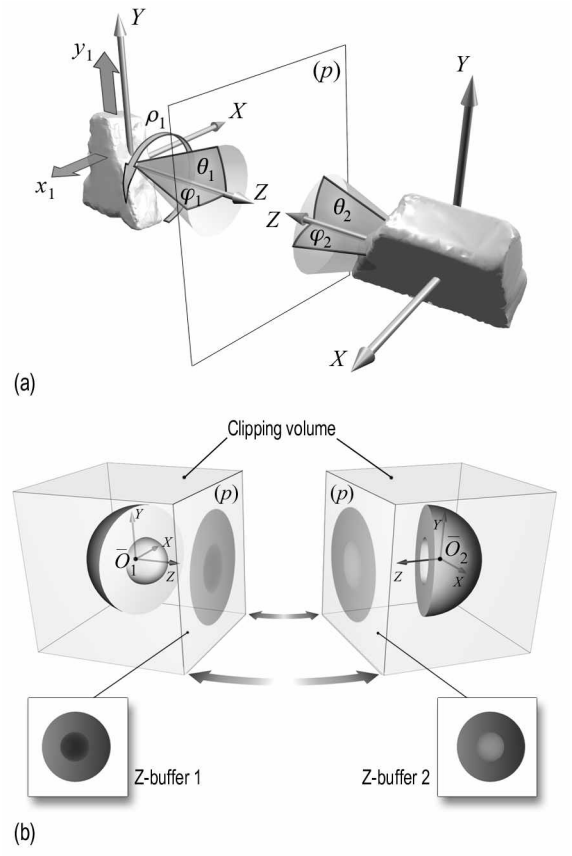


Distance Function of a Point Site
Figure 4

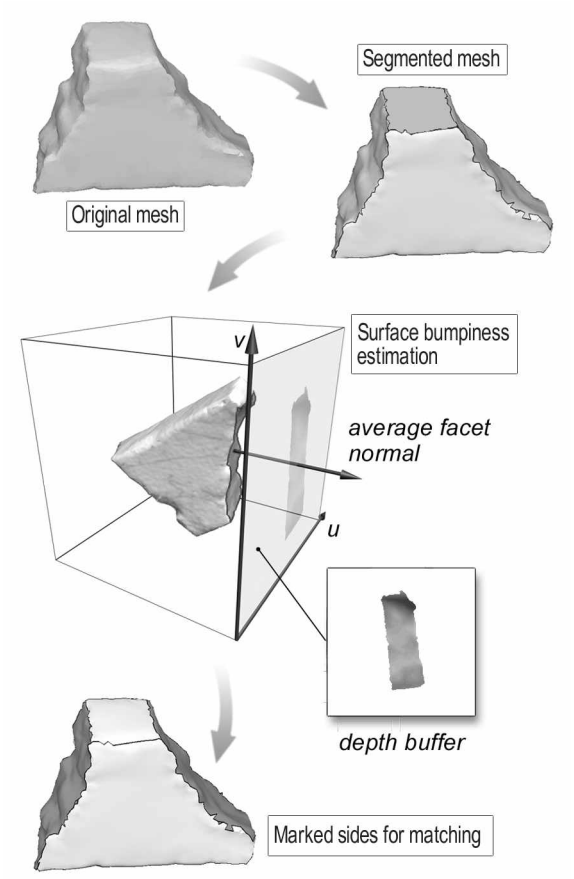
The complementary matching is based on the minimisation of a point to point distance-based matching error between the facing surface segments of two fragments. As the method must be able to handle objects of arbitrary shape and mesh topology, the matching error estimation is image-based and utilises the Z-buffer for the uniformly sampled point-to-point distance between the fragments. This matching error is minimized, employing a standard global optimization scheme, to determine the relative positioning of the two fragments that corresponds to their best complementary fit.

More specifically, the two fragments are positioned in a way that two of their broken facets are facing each other (Fig. 5a). A set of seven pose parameters is adequate for the alignment of the two fragments. The first object is allowed to perform a full circle around the axis of alignment (r_1), deviate from this axis (f_1,q_1) by up to 10^o and slide along the broken facet (x_1, y_1). The second object need only diverge from the axis of alignment by up to 10^o (f_2,q_2).

The matching error calculation uses the derivatives of the point-to-point distance between the facing sides of the fragments (surface curvature). These distances are evaluated by rendering each object using as viewing plane a separating plane (*p*) between the objects (Fig 5b). If $Z_1 = Z(\bar{O}_1, \vec{X}_{(obj1)}, \vec{Y}_{(obj1)}, -\vec{Z}_{(obj1)}, -R, R, LESS)$ and $Z_2 = Z(\bar{O}_2, -\vec{X}_{(obj2)}, \vec{Y}_{(obj2)}, -\vec{Z}_{(obj2)}, -R, R, LESS)$ where *R* is the maximum radius of both objects, the matching error e_d is given by:



Complementary matching between 2 object fragments using the Z-buffer.
Figure 5



Detection of fractured faces on an object.
Figure 6

$$e_d = \frac{1}{A_s}\iint_S \left(\left| \frac{\partial Z_1(u,v)}{\partial u} + \frac{\partial Z_2(u,v)}{\partial u} \right| + \left| \frac{\partial Z_1(u,v)}{\partial v} + \frac{\partial Z_2(u,v)}{\partial v} \right| \right) dS$$

where S is the buffer region where the two facets overlap and A_s is the corresponding area of overlap.

Ideally, if the broken surfaces of the fragments are complementary, the matching error should be zero for a relative pose of the two pieces where they "fit" together. For all other placement configurations or for incompatible fragments, there is a significant matching error between the fragment facets.

This application is one of the rare cases where rendering is not used just for visualization, but actively participates in the solution of a computer vision problem.

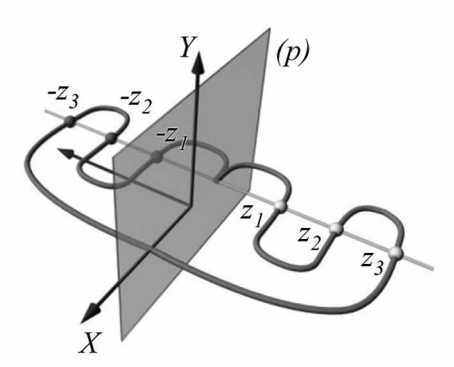
In order to define which sides of the fragments to test for complementary matching, the following method is used [Papai00b]: A fragment's mesh is first partitioned into regions of nearly coplanar polygons (facets) by a region growing and merging scheme. Then, those facets that exhibit higher surface irregularity are assumed to be fractured and are labeled as potential for matching.

A measure of the facet's irregularity can not be estimated directly from the original mesh (unless the surface is uniformly sampled) because each facet consists of polygons of arbitrary connectivity and varying area. Instead, an image-based bumpiness measure is calculated on the *elevation map* of the facet. Obviously, the elevation map corresponds to the Z-buffer if the facet triangles are rendered with the viewing direction parallel to the average facet normal (Fig. 6).

The bumpiness of a surface is associated with the rate of elevation variance and can be effectively estimated on the elevation map with an image filter, such as the Laplace image operator.

8. SYMMETRY DETECTION

Symmetry of three-dimensional objects is a valuable property that is of use in a wide range of applications, e.g. object recognition and



Reflectional symmetry detection.
Figure 7

Rotational symmetry detection.
Figure 8

reconstruction from views, or data compression.

Reflectional and rotational symmetry of a 3D object can be measured using a modified version of the traditional depth buffer, where all depth values are stored, instead of the minimum ones [Karab00]. For every pixel (x, y), the modified depth buffer holds all corresponding depth values, in ascending order. This is modelled by introducing a new depth buffer operator *ALL*.

A 3D object has reflectional symmetry if it is invariant under reflection about a plane, which crosses the object's centre of mass. If *(p)* is a candidate reflectional symmetry plane for a given object, the viewing plane is placed to coincide with *(p)* (as in Fig. 7). Without loss of generality, let this plane be $z = 0$. If *(p)* is indeed a plane of symmetry, every object point which lies on one side of *(p)* must have a counterpart on the other side, with identical (x, y) coordinates and opposite *z* values. Any difference in the number of discrete object points encountered at each side of *(p)*, or in the *z* values that correspond to a given (x, y), leads to a deviation from perfect symmetry.

If a modified depth buffer $AZ = Z\left(\bar{C}_Z, \vec{X}, \vec{Y}, -\vec{Z}, z_{\min}, z_{\max}, ALL\right)$ is generated, the above conditions can be checked simply by examining the z-buffer values for each pixel (x, y). Let $AZ(x, y)_i, i = 1...N(x, y)$ represent the i-th buffer value at (x, y). Let also $N_+(x, y)$ and $N_-(x, y)$ be the number of positive and negative z-buffer values at (x, y), respectively. The object is non-symmetrical at (x, y), if :

$$N_+(x, y) \neq N_-(x, y)$$

or

$$\exists i \in 1...N_+(x, y) : \left| AZ(x, y)_i + AZ(x, y)_{N(x,y)-i} \right| > e$$

where $e \in [0,1]$ defines the algorithm tolerance.

A 3D object has rotational symmetry of order *n* if it is invariant under rotation of $2p/n$ radians about an axis passing through the object's centre of mass. Rotational symmetry of order *n* can be detected using *n* depth buffers (Fig. 8), placed perpendicular to the candidate symmetry axis *(l)*. Subsequent buffers form a $2p/n$ radians angle. In this case, instead of comparing positive and negative depth values in a single buffer, only positive values are required; positive depth values for each point (x, y) are compared between two subsequent z-buffers. Once a symmetry error is determined for all *n* pairs of buffers, its average value defines the overall symmetry error.

The symmetry measures described in this section can be used to as error functions in a global optimisation scheme, to locate the actual planes/axes of symmetry.

A similar approach can be adopted for congruity detection, where depth buffers of two different objects are compared. In both cases of symmetry and congruity detection, a "traditional" depth buffer can be used, if the objects under examination are convex.

9. CONCLUSION

The Z-buffer has turned out to be a much more powerful tool than it was originally intended. We have presented a survey of important and recent applications. No doubt this list will continue to grow.

REFERENCES

[Catmu74] Catmull,E.: A Subdivision Algorithm for Computer Display of Curved Surfaces, *PhD Thesis*, Dept of Computer Science, University of Utah, Salt Lake City, Utah, U.S.A., 1974.

[CGL00] http://graphics.di.uoa.gr/~graphics

[Cook86] Cook,R.L.: Stochastic Sampling in Computer Graphics, *Transactions on Graphics, ACM*, Vol.5, No.1, pp. 51-72. 1986.

[Duff85] Duff,T.: Compositing 3D Rendered Images, *ACM Computer Graphics*, 19(3), SIGGRAPH 1985, pp. 41-44.

[Epste89] Epstein,D., Jansen,F., Rossignac,J.: Z-Buffer Rendering from CSG: The Trickle Algorithm, *IBM Research Report RC15182*, 1989.

[Foley91] Foley,J.D., van Dam,A., FeinerS.K., Hughes,J.F.: *Computer Graphics, Principles and Practice, Addison-Wesley*, 1991.

[Goldf86] Goldfeather,J., Hultquist,J., Fuchs,H.: Fast Constructive Solid Geometry in the Pixel-Powers Graphics System, *Computer Graphics (SIGGRAPH '86 Proceedings), ACM*, Vol.20, No.4, pp. 107-116, 1986.

[Hoff99] Hoff,K.E., Culver,T., Keyser,J., Lin, M., Manocha,D.: Fast Computation of Generalised Voronoi Diagrams Using Graphics Hardware, *ACM Computer Graphics*, SIGGRAPH 1999, pp. 277-285.

[Karab99] Karabassi,E.A., Papaioannou,G., Theoharis,T.: A Fast Depth-Buffer-Based Voxelization Algorithm, *Journal of Graphics Tools*, *ACM*, Vol.4, No.4, pp.5-10, 1999.

[Karab00] Karabassi,E.A., Papaioannou,G., Theoharis,T.: Accurate and Fault-Tolerant Detection of Symmetries in 3D Objects, submitted for publication.

[Papai00a] Papaioannou,G., Karabassi,E.A., Theoharis,T.: Automatic Reconstruction of Archaeological Finds – A Graphics Approach, *International Conference on Computer Graphics and Artificial Intelligence* (*3IA 2000 Proceedings*), pp. 117-125, 2000.

[Papai00b] Papaioannou,G., Karabassi,E.A., Theoharis,T.: Segmentation and Surface Characterization of Arbitrary 3D Meshes for Object Reconstruction and Recognition, *International Conference on Pattern Recognition (ICPR '2000 Proceedings), IEEE*, 2000.

[Porte84] Porter,T., Duff,T.: Compositing Digital Images, ACM Computer Graphics, 18(3), SIGGRAPH 1984, pp. 253-259.

[Reeve87] Reeves,W.T., Salesin,D.H, Cook,R.L, Rendering Antialiased Shadows with Depth Maps, *Computer Graphics (SIGGRAPH '87 Proceedings), ACM*, Vol.21, No.4, pp. 283-291, 1987.

[Segal99] Segal,M., Akeley,K., Eds: Frazier,C., Leech,J.: *The OpenGL Graphics System: A Specification* (v. 1.2.1), 1999, Silicon Graphics Inc.

[Stewa98] Stewart,N., Leach,G.: An Improved Z-Buffer CSG Rendering Algorithm, *1998 Eurographics / SIGGRAPH Workshop on Graphics Hardware Proccedings, ACM*, pp. 25-30, 1998.

[Stewa00] Stewart,N., Leach,G., John,S.: A Z-Buffer CSG Rendering Algorithm for Convex Objects, *8-th International Conference in Central Europe on Computer Graphics (WSCG '2000 Proceedings)*, 2000.

[Voron08] Voronoi,G.M.: Nouvelles Applications des Parametres Continus a la Theorie des Formes Quadratiques. Deuxieme Memoire: Recherches sur les Parallelloedres Primitifs, *J. Reine Angew. Math.*, 134, 1908, pp. 198-287.

[Wiega96] Wiegand,T.F.: Interactive Rendering of CSG Models, *Computer Graphics Forum, Blackwell Publishers*, Vol.15, No.4, pp. 249-261, 1996.

[Willi78] Williams,L.: Casting Curved Shadows on Curved Surfaces, *Computer Graphics, ACM*, Vol.12, No.3, pp. 270-274, 1978.