Computer Science Department of The University of Auckland
CITR at Tamaki Campus (http://www.citr.auckland.ac.nz)

CITR-TR-69 September 2000

A Global Surface Area Estimation Algorithm for Digital Regular Solids

Reinhard Klette[1] and Hao Jie Sun

Abstract

The paper estimates the surface area of regular solids by a global segmentation of a digital representation of the given solid into digital plane segments, and the projection of these digital plane segments into Euclidean planes. Multigrid convergence experiments of the estimated surface area value are used to evaluate the performance of this new method for surface area measurement. multigrid convergence.

[1] Centre for Image Technology and Robotics, The University of Auckland, Tamaki Campus, Auckland, New Zealand

A Global Surface Area Estimation Algorithm for Digital Regular Solids

Reinhard Klette and Hao Jie Sun

CITR Tamaki, University of Auckland
Tamaki Campus, Building 731, Auckland, New Zealand

Abstract. The paper estimates the surface area of regular solids by a global segmentation of a digital representation of the given solid into digital plane segments, and the projection of these digital plane segments into Euclidean planes. Multigrid convergence experiments of the estimated surface area value are used to evaluate the performance of this new method for surface area measurement.

1 Introduction

Gridding techniques are widely used to represent volume data sets in three-dimensional space in the field of computer-based image analysis. The problem of multigrid convergent surface area measurement had been discussed for more than one hundred years [10] and not yet reached a satisfactory result. In [6], *regular solids* are defined as the models of '3D objects' and as sets in $\mathcal{R}^3$ being candidates for multigrid surface area studies. The surface areas of such sets are well defined.

C.Jordan [5] studied the problem of volume estimation based on gridding techniques in 1892, and C.F.Gauss (1777-1855) studied the area problem for planar regions also based on this technique. Gridding is in today's point of view also considered as digitization which maps a 'real object' into a grid point set with certain resolution r defined as being the number of grid points per unit. *Jordan digitization* is characterized by *inclusion digitization*, being the union of all cubes completely contained in the topological interior of the given regular solid, and *intersection digitization*, being the union of all cubes having a non-empty intersection with the given set. *Gauss center point digitization* is the union of all cubes having a center point in the given set. The Gauss center point digitization scheme is used in this paper. It maps given regular solids into orthogonal grid sets, in which each grid edge (i.e. an edge of a grid square or grid cube) is of uniform length (grid constant).

The effect of studying objects enlarged r times considering the grid constant as 1 is the same as studying the object of original size having resolution r (or grid constant $1/r$). The advantages of the former (preferred by Jordan and Minkowski) are that calculations of surface areas are integer arithmetic operations and it is chosen in our implementation later illustrated. This is also called the general *duality principle for multigrid studies*.

The paper [6] proposed a way to classify polyhedrization schemes for surfaces of digitized regular solids based on the notion of *balls of influence* $B(p)$. Let $D_r(\theta)$ be the digitized set of a regular solid θ with a grid resolution r. For each polygon $\mathbf{G}$ of the constructed polyhedron there is a subset of $D_r(\theta)$ such that only grid points in the subset have influence on the specification of polygon $\mathbf{G}$. This subset is the ball of influence. Due to the finite number of calculated polygons, there is a maximum value $R(r, \Theta)$ of all radii of the ball of influence. The polyhedrization techniques are then classified based on the following criterion:

Definition 1. *A polyhedrization technique is* local *if there exists a constant* κ *such that* $R(r, \Theta) \leq \kappa/r$*, for any regular solid* Θ *and any grid resolution* r*. If a polyhedrization method is not local then it is* global*.*

For the surface detection algorithm of [1], the marching cubes algorithm and marching tetrahedra algorithm, the constant κ is not more than $\sqrt{3}/2$ [8].

The soundness of a grid technique as pointed out by [7] should meet in short the following properties with respect to surface area estimation:

1. Higher resolutions should lead to convergence for the value of surface area.
2. Convergence should be towards the true value.

Obviously, surface area calculation by counting the *surfels* (grid faces of the digital surface) of the digitized regular solid is not meeting the criteria, since the result of doing this may converge to a value from 1 to $\sqrt{3}$ times the true value depending on the shape and position of the given object when the resolution r goes to infinity [9]. Other local polyhedrization techniques as investigated in [6] such as marching cube and dividing cube algorithms, although they all converge, fail to converge to the true value.

On the other hand, a proposed global technique, minimum surface area polyhedrization [11] which is implemented for convex sets by the calculation of the convex hull, converges to the true value when dealing with these convex sets [6]. Our method is also a global technique. In this method the surface of the given digital object is first segmented by agglomerating surfels into maximum digital plane segments (DPSes). Then the sum of the surface areas of projections of these DPSes is calculated and used to estimate the surface area of the digitized object. Our global surface area estimation technique is called *DPS method*.

Section 2 gives our definition of a digital plane segment. Section 3 explains the algorithm to recognize a digital plane segment, and to perform the surface area estimation. The experimental results are presented and discussed in Section 4. We conclude in Section 5. In this paper, we restrict our interest on implementation and experimental results. The theoretical convergence analysis is left to another paper.

2 Digital Plane Segments

We generalize from a definition of a digital straight segment (DSS) in the plane as introduced in [4]. Two parallel lines having a given 4-path in between are called *bounding lines*. There are two possible diagonals in 2D grid space, see Fig. 1.

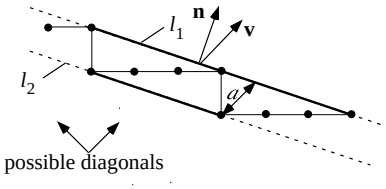

Fig. 1. Definition of a DSS based on main diagonals: l_1 is the line containing the maximum length tangential line segment of the 4-curve, l_2 is the second tangential line parallel to l_1, and $a < \sqrt{2}$ is the main diagonal distance between both. Vector $\boldsymbol{n}$ is the normal of l_1, and $\boldsymbol{v}$ is the unit vector of the main diagonal.

The *main diagonal* for a pair of parallel lines is the one which maximizes the dot product with the normal of these lines. The *main diagonal distance* between these parallel lines is measured in direction of the main diagonal. A 4-path is a DSS iff there is a pair of bounding lines having a main diagonal distance less than $\sqrt{2}$.

We will define a digital plane segment (DPS) in 3D space by a main diagonal distance between two tangential planes. For a finite set of faces of grid cubes in 3D space (being a subset of a digital surface) we consider a slice defined by two parallel planes. One plane is tangential to the face vertices of this set, and the other (parallel) plane is touching the vertices of this set 'from the other side'. Fig.2 gives a rough illustration of a DPS. Again, $\boldsymbol{n}$ denotes the normal, and $\boldsymbol{v}$ is the unit vector in direction of the main diagonal.

A grid cube has eight directed diagonals. The *main diagonal* of a pair of parallel planes is that diagonal direction out of these eight directed diagonals which has the maximum dot product (inner product) with the normal of these parallel planes. Note that there may be more than one main diagonal direction for a pair of planes, and we can choose any of these as our main diagonal. The distance between two parallel planes in main diagonal direction is called the *main diagonal distance* of these two planes.

Definition 2. *A finite set of grid faces in 3D space, being edge-connected, is a DPS iff there is a pair of parallel planes containing this set in-between, and having a main diagonal distance less than* $\sqrt{3}$.

It follows that the projection of a DPS on any of the coordinate planes (i.e. $X = 0$, $Y = 0$ or $Z = 0$) has no multiple points: given the two parallel planes and the boundary of the projection of the DPS it is possible to reconstruct the DPS in 3D space.

Let $\boldsymbol{v}$ be a vector in a main diagonal direction with length of $\sqrt{3}$, $\boldsymbol{n}$ be the normal vector of the pair of planes and $d = \boldsymbol{n} \cdot \boldsymbol{p}$ be the equation for one of these two planes. According to Def. 2, all the vertices p of the grid cubes of a DPS must satisfy the following inequality,

$$0 \leq \boldsymbol{n} \cdot \boldsymbol{p} - d < \boldsymbol{n} \cdot \boldsymbol{v} \tag{1}$$

Let $\boldsymbol{n} = (a, b, c)$. The scalars a, b, c may have different signs, but due to the consideration of $\boldsymbol{n} \cdot \boldsymbol{v}$ we can assume that $a, b, c > 0$ w.l.o.g. Then this inequality

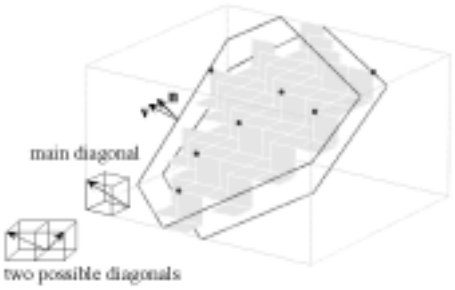

Fig. 2. A 3D view of a DPS: the main diagonal distance between the two bounding planes is less than $\sqrt{3}$.

becomes

$$0 \leq ax + by + cz - d < a + b + c \ , \tag{2}$$

i.e. our DPS definition is equivalent to that of a finite, edge-connected subset of faces in a standard plane [2]. A *simply-connected DPS* is such that the union of its faces is topologically equivalent (in Euclidean space) to the unit disk.

3 The Surface Area Estimation Algorithm

The surface of the digitized regular solid consists of grid faces, and these may be traced by using [1]'s surface tracking algorithm to build a surface graph. Each node of the surface graph represents a grid face as well as four pointers to all of its four adjacent faces. By this representation, we can implement a breadth-first search of all the faces, to agglomerate faces into segments such that all faces in one segment belong to one DPS. We are interested in calculating maximum sets of faces satisfying the DPS definition.

The approach to agglomerate faces according to (1) is a classical problem which is also known as recognition of a digital plane segment in computer imagery. The problem can be represented as follows: given n points $\{p_1, p_2, \ldots, p_n\}$, does there exists a DPS such that each point satisfies the inequality (1), i.e. is it possible to solve the following inequality system:

$$0 \leq \boldsymbol{n} \cdot \boldsymbol{p}_i - d < \boldsymbol{n} \cdot \boldsymbol{v}, \quad i = 1, \ldots, n \tag{3}$$

This inequality system contains four scalar unknowns, d and $\boldsymbol{n} = (a, b, c)$. By eliminating d, we get a new inequality system Equ.4 which has n^2 inequalities:

$$\boldsymbol{n} \cdot \boldsymbol{p}_i - \boldsymbol{n} \cdot \boldsymbol{p}_j < \boldsymbol{n} \cdot \boldsymbol{v}, \quad i, j = 1, \ldots, n \tag{4}$$

This inequality system no longer requires $\boldsymbol{n}$ to be a normal vector, thus it is turned into a linear homogeneous inequality system with three unknowns a, b, c and it can be solved in various ways. For example, the paper [2] presents a Fourier

```
// breath search to traverse the isosurface to build a face graph
// argument: inif - initial face
// return: a container holding the faces of the face graph
vector<Face> face_tracking(Face initf) {
  vector<Face> faces;               // container to keep the face graph
  queue q;
  q.enqueue(initf);                 // Put initial face into the queue
  while (!q.empty()) {
    Face f = q.dequeue();           // breath first search
                                    // for adjacent faces
    faces.add(f);
    for(int i=0; i<4; i++) {        // loop each of the 4 adjacent faces
      Face adjf = f.adjacentFace(i);     // the ith adjacent faces
      if (!find(faces, adjf)) {     // if not already added to the graph
        if (!find(queue, adjf))     // and not in the queue
          q.enqueue(adjf);
        f->setNeighbor(adjf);       // set pointer to adjacent face
      }
    }
  }
  return faces;
}
```

Fig. 3. Breadth-first search to construct a face graph.

elimination approach. Computationally this turned out (in our experiments) to be not time-efficient. More efficient algorithms include operation research's linear programming, which is to find an optimal solution given an object function and a set of linear constraints. If such a solution exists, the inequality system is solved. Quick Hull [12] is another method which constructs a convex hull by cutting the 3D space given a finite set of half-spaces. By finding out whether a non-empty convex hull exists, the inequality system is then solved. CDD [13] implements the Double Description Method of Motzkin et al. for generating all vertices, i.e. extreme points and extreme rays of a general convex polyhedron in R^d given by a system of linear inequalities. Since in larger grid resolutions, the number of inequalities may be very large, it is critical to have an elegant and efficient algorithm to solve the inequality system. We have experimented all of the above mentioned methods and found out that the CDD algorithm works best and fastest in our implementation. Therefore in this paper, we use the CDD to solve our n^2 inequalities.

We start the DPS recognition process from a face graph which is obtained by adapting [1]'s surface tracking algorithm. Each node of the face graph is a grid face of the surface of the digitized set with four pointers to its four adjacent neighbors. The pseudocode is listed in Fig. 3.

After the face graph is constructed, we start another breadth-first search in the face graph to agglomerate the faces into DPSes. This breadth-first search algorithm is implemented using two queues. One is called a seeds queue, containing all searched faces not yet belonging to any recognized DPS. Whenever

```
// polyhedrize a regular set given its isosurface face graph
// arguments: inif - a node of the surface graph to start from
// return: a container of patches, each patch itself is a container
//         for holding faces
vector<Patch> polyhedrise(Face initf) {
  vector<Patch> patches;      // container holding DPSes
  queue q;                    // queue to maintain
                              // the breath first search
  queue seeds;                // contains faces to start a new DPS
  seeds.equeue(initf);
  while (!seeds.empty()) {
    Face s = seeds.dequeue();
    if (s.isAssigned)         // if s is already assigned to another DPS
      continue;
    Patch* p = new Patch();   // container to hold the faces
                              // belonging to a DPS
    q.enqueue(s);             // start breath first search
                              // to recognise a DPS
    while (!q.empty()) {
      Face f = q.dequeue();
      if (!putFace(p, f)) {   // test: f can be added to the current
                              // patch? - if so, then add it
        if (!find(seeds, f)   // failed, f is already in queue 'seeds'?
          seeds.enqueue(f);   // no, add it to seeds queue
          continue;
        }                     // added successfully,
                              // expand its adjacent faces
        for (int i=0; i<4; i++) {
          Face adjf = f.adjacent(i);
          if (!adjf.isAssigned() && !find(q, adjf))
                              // if not assigned and not in q
              q.enqueue(adjf);
        }
    }
    patches.add(p);
  }
  return patches;
}
```

Fig. 4. Breath-first search for recognizing a DPS.

a face cannot be added to the current DPS, it is inserted into the seeds queue. The next DPS will start from one face chosen from the seeds queue. The second queue is used to maintain the breath-first search, so that growing of the DPS looks like the propagation of a 'circular wave'. The breath-first search algorithm is listed in Fig. 4.

When an adjacent face is found, we try to add it to the current DPS. Therefore we create an instance of the inequality system and solve it using the CDD algorithm. If a feasible solution is found, we add this face to the current DPS and delete it from the seeds queue if it is also there. Otherwise we insert this

```
// test if a face f belongs to a DPS p
bool putFace(Patch p, Face f) {
  switch (number of p's directions) {
  //face with the first direction and second direction must belong to p
  case 2:    //the third direction is opposite to an existing direction
    if (f is opposite to one of p's directions or !cddsolve(p, f))
      return false;
    break;
  case 3:    //no forth direction is allowed
    if (f is not one of p's directions or !cddsolve(p, f))
      return false;
  }
  p.add(f);
  if (f is a new direction to p)
      add f's direction to p's direction list
  return true;
}
```

Fig. 5. Test whether a face belongs to a DPS.

face into the seeds queue and try another adjacent face. If no further adjacent face can be added, we start a new DPS from a face in the seeds queue. Realizing that only three face directions are allowed in one DPS, this constraint is added to the algorithm so that the program can be sped up a bit. Another observation is that no two opposite face directions are allowed in one DPS. This process is illustrated in Fig.5. The *cddsolve* function in Fig.5 is to create an instance of the inequality system and solve it by the CDD program, which is straightforward and we will not discuss it. When adding a face, all of its four vertices must be considered. Except those that are already in the current DPS, the remaining vertices must be added to the inequality system. The CDD algorithm will give a result showing whether the inequality system is feasible.

As a result, the surface of the digitized regular solid is composed of DPSes, which are in turn composed of faces. The faces of one DPS are not coplanar in the sense of Euclidean geometry. To evaluate the surface area of a DPS, we must first project the surfels on one of the bounding planes of the DPS and then sum up the projected area. The surface area of the original regular solid is then estimated by the sum of areas of all DPSes. Of course, calculating a polyhedron first using all the calculated bounding planes, and using the surface area of such a polyhedron as an estimator might be a way to produce a more accurate value. But the experimental results reported below show that the given method is very precise already.

4 Experimental Results

The calculation of the surface area of an ellipsoid with all three semi-axes a, b, c being allowed to be different is known to be a complicated task. If two radii coincide, i.e. in case of an ellipsoid of revolution, the surface area can by analytically specified in terms of standard functions. The surface area formula in the

general case is based on standard elliptic integrals. Example 2 in [8] (reporting recent work done by Garry Tee) specifies an analytical method to estimate the surface of such ellipsoids. The value calculated by Garry Tee's program for calculating this surface area is used in this paper as 'ground truth' to evaluate the performance of our DPS algorithm.

Figure 6 shows a digitized sphere and ellipsoid where faces are grouped into DPSes. The search depth is restricted to 7 for efficiency and optimization consideration. For large resolution, such a search depth restriction is actually degrading the global technique into a local one. The total numbers of faces of the shown digitized sphere and ellipsoid are 7584 and 4744, respectively, at resolution r=40. The number of DPSes of the sphere and ellipsoid are 247 and 160, respectively. The average size of one DPS is approximately 30 faces in both cases.

The surface area of the ellipsoid is estimated where resolution increases from 10 to 100. Three positions of the ellipsoid are investigated. The left diagram in Fig. 7 shows results for these three different positions. It illustrates that the DPS method converges rather fast and not continuously (with increasing grid resolution) to the true value. This is due to the face that with different resolution and initial faces the resulting segmentation may differ. Compared to the convex hull or marching cubes algorithm at the same resolutions, the accuracy of our DPS method is far better than the other two at the same resolution. The convex hull method and marching cube have a relative error of 3.22% and 10.80% at r=100, respectively, while the DPS error is less than 0.4%. Note that this result is a bit arbitrary, due to the dependence on initial face and search strategy.

The right figure of Fig. 7 shows estimated surface areas when the resolution is fixed at 20, 60 and 100, and one of the semi-axis of the ellipsoid changes from 2 to 20. Different from the convex hull method, the relative errors remain nearly constant as the ellipsoid goes from a 'thin board' to a sphere.

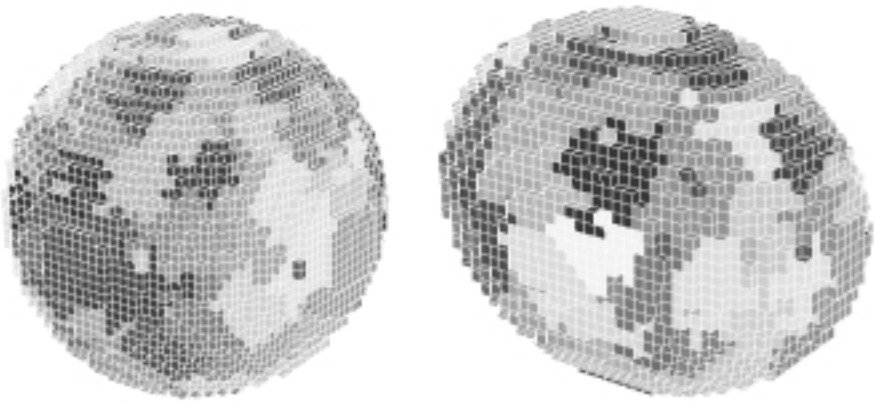

Fig. 6. Surfaces of a sphere and a 20*16*12 ellipsoid digitized at $r = 40$ segmented into DPSes with a face search depth limited to 7.

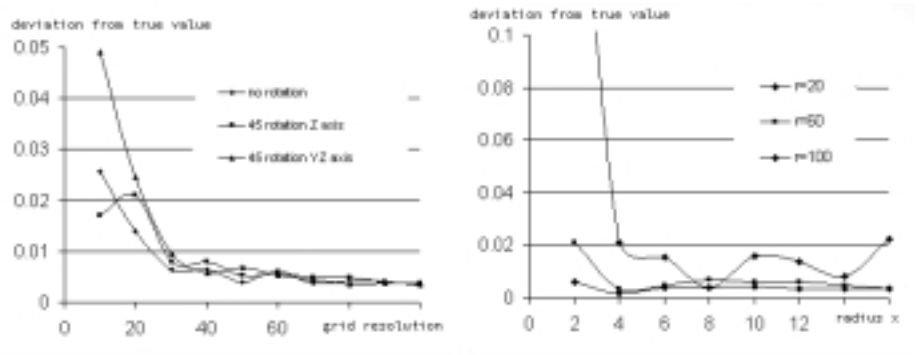

Fig. 7. Left: surface area estimation of a 20*16*12 ellipsoid at different resolutions and positions. Right: surface areas estimation of a 20*16*x ellipsoid at resolution r = 20, 60, 100 where x changes from 2 to 12.

An impossible situation for the convex hull method is given when non-convex solids have to be considered. We also use non-convex regular solids defined by an ellipsoid having an 'inner tangential ellipsoidal hole'. Half of such a solid is visualized on the left of Fig. 8. Results for three positions are shown on the right of Fig. 8. For the non-rotated position the relative error is 0.17% when the resolution is 100. For rotated positions, the relative errors are less than 0.03%. The phenomenon that a non-convex object has a smaller error than a convex object can be explained by the reason that the hole is actually compensating a part of the error caused by the digitization scheme.

5 Conclusion

We designed and implemented a global surface area estimation algorithm for digital surfaces. Compared to other techniques such as based on marching cubes isosurfaces or convex hull calculations, our DPS method converges faster and

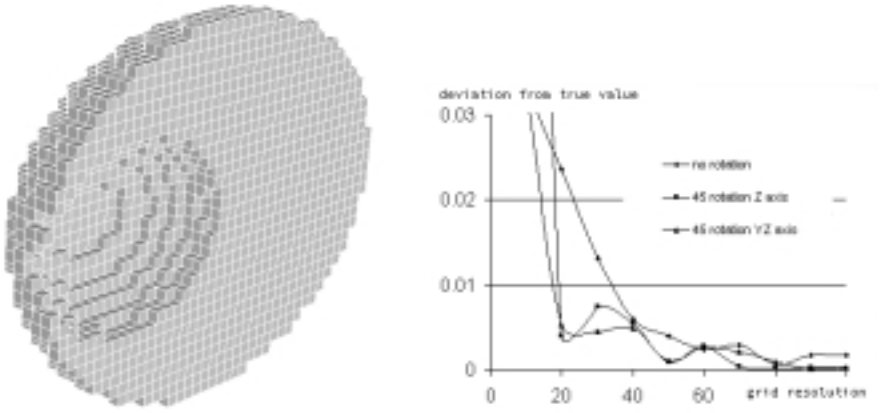

Fig. 8. Left: Cut through a non-convex solid. Right: surface area estimation of a non-convex solid at three different positions

seems to converge to the true value. When the algorithm is applied to non-convex sets, our result is even more accurate than the convex hull method. Theoretical analysis of convergence to the true value (for specific convex solids) is left to a forthcoming paper. Also the classic problem of recognizing DPSes lacks an efficient and less memory consuming and practical algorithm. Although the algorithm chosen in this paper to decide whether a face could be added to a DPS or not is a great improvement in efficiency compared to [2]'s Fourier elimination algorithm, an incremental way to solve the inequalities system should be possible and could improve the current algorithm. With a more efficient algorithm, larger resolutions are assumed to better illustrate the convergence behavior of our DPS method. It will also be interesting to study the convergence behavior w.r.t. different search depth restrictions.

Acknowledgements

Thanks are due to Garry Tee (Auckland), Laurent Papier (Strasbourg), Yukiko Kenmochi (JAIST), and Feng Wu (Auckland).

References

1. E. Artzy, G. Frieder, and G. T. Herman: The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm. *CVGIP* **15** (1981) 1–24.
2. J. Françon, J.-M. Schramm and M.Tajjine: Recognizing arithmetic straight lines and planes. Proc. Int. Workshop on *Discrete Geometry for Computer Imagery (6th DGCI Conference)*, **LNCS 1176**, Springer, Berlin Heidelberg (1996) 141–150.
3. J. Françon and L. Papier: Polyhedrization of the boundary of a voxel object. *DGCI'99*, **LNCS 1568**, Springer, Berlin Heidelberg (1999) 425–434.
4. H. Freeman: Boundary encoding and processing. in: B.S. Lipkin and A. Rosenfeld, eds., *Picture Processing and Psychopictorics*, Academic Press (1970) 241–266.
5. C. Jordan: Remarques sur les intégrales définies. *Journal de Mathématiques* 4^e **série** (1892) T. 8, 69–99.
6. Y. Kenmochi and R. Klette: Surface area calculation for digitized regular solids. *SPIE's 45th Annual Conf., 4117 (Vision Geometry IX)* 30 July - 4 August 2000, San Diego, to appear.
7. R. Klette: Approximation and representation of 3D objects. in: *Advances in Digital and Computational Geometry*, Springer, Singapore (1998) 161–194.
8. R. Klette and B. Yip: The length of digital curves. *Machine GRAPHICS & VISION* **9** (2000) to appear.
9. A. Lenoir, R. Malgouyres, and R. Revenu: Fast computation of the normal vector field of the surface of a 3-d discrete object. *Discrete Geometry for Computer Imagery* (S. Miguet, A. Montanvert, S. Ubéda, eds.) **LNCS 1176**, Berlin, Springer (1999) 101–112.
10. H. A. Schwarz: Sur une définition erronée de l'aire d'une surface courbe, *Ges. math. Abhandl.* **2** (1890) 309–311.
11. F. Sloboda, B. Zatko and R. Klette: On the topology of grid continua. *SPIE's 43rd Annual Meeting*, 19-24 July (1998), San Diego, Conference **3454** (Vision Geometry VII), 52–63.
12. http://www.geom.umn.edu/software/qhull/
13. http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html