Deriving shape from shadows. A Hilbert space setting.

MICHAEL HATZITHEODOROU

Department of Computer Science
Columbia University
New York, NY 10027

CUCS - 288 - 87

November 1987

Abstract. We study the problem of recovering a surface slice from the shadows it casts on itself when lighted by the sun at various times of the day. The problem is formulated and solved in a Hilbert space setting. The spline algorithm interpolating the data that result from the shadows is constructed. This algorithm is optimal in terms of the approximation error and has low cost. We implement the optimal error algorithm and show a series of test runs. In addition, another modified version of the algorithm that improves the cost considerably is shown. This version is suited for parallel computation with further reductions in the cost of the solution.

1. Introduction.

In computer vision the surface reconstruction problem is of crucial importance in the process of recovering the scene characteristics from the 2-dimensional image created by the camera.

In the various *Shape from X* algorithms, different methods of recovering a surface have been proposed. In this paper we will use a new approach for the reconstruction of the surface shape. Namely, we will define and solve the *Shape from Shadows* problem. In this problem the shadows created by a light falling on a surface will be used to recover the surface itself.

Very little work has been done using shadows for the reconstruction of the surface shape. The only work we are aware of is the one proposed in [5] where the problem is solved using a relaxation method.

Shadows are a very strong piece of information. The process that uses shadows is not affected by texture or by surface reflectance. Furthermore, our imaging system does not need a grey scale or color capabilities; it is sufficient for it to be able to distinguish between black and white. Also, noise in the form of bright spots inside a dark area and vice-versa can be filtered out easily. From the above, it is evident that shadows yield a powerful tool to be used in the reconstruction process.

Our problem will be the following. We have a surface which is lighted by a light source. The light source casts shadows on the surface. Then the light moves to a new position where

it casts new shadows. We collect the different images, of the shadowed surfaces, at these various times. From those we obtain the location of the start and the end of the shadow, plus some additional information about the surface function. Given any series of images containing shadows, we want to obtain an algorithm that produces an approximation to the surface with the smallest possible error.

We choose in this approach to recover slices of the surface. A slice is defined as the intersection of the surface and a plane of constant y (Fig. 1.) We assume that the surface slice is a function defined on the interval $[0, 1]$, having continuous first derivatives, and second derivatives with bounded L_2 norm.

We propose an algorithm that recovers the surface and minimizes the worst possible error within a factor of 2. The algorithm that minimizes the error is the spline algorithm.[1] We choose to construct the spline algorithm in steps using a process that converges to the optimal error algorithm. The justification behind this stepwise construction is that, in general, the optimal error algorithm might need many iterations to construct, but in most practical cases the initial version of the algorithm, or the one resulting from a few iterations, already obtains the optimal error. We therefore construct a process that has low cost, while achieving the smallest possible error.

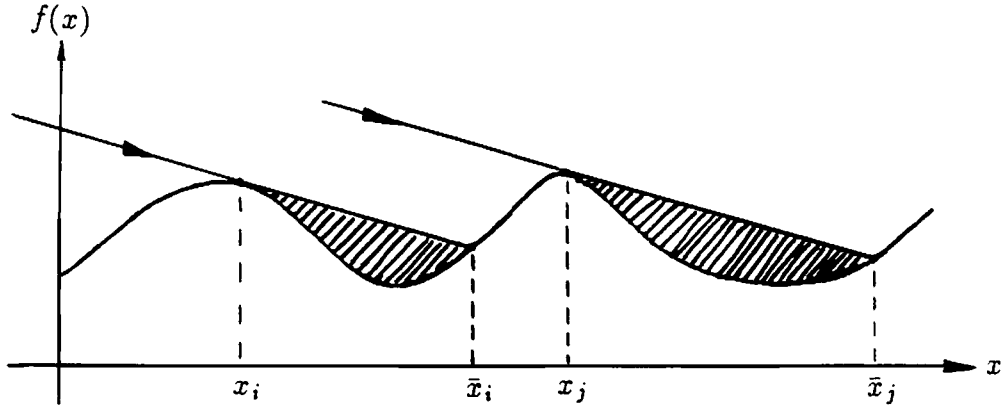

FIGURE 1.

We have done a series of numerical runs to test the performance of the above process. The obtained approximations were very close to the function from which we obtained the data, and that can be immediately seen from the pairs of initial and its reconstruction that we are supplying. We also propose a parallel implementation of the algorithm that will considerably improve the running time.

The organization of the rest of the paper will be the following : In section 2, we will formulate the problem; we will define a function space in which our surface must belong. We will also define more precisely the information that can be extracted from the shadows.

In section 3, we will define the optimal error algorithm. We show that this is the spline algorithm which always exists and is unique. This will guarantee that the shape from shadows problem under this formulation is well-posed. The definition of a well-posed problem can be found in [3, 10]. In contrast to many other *shape from X* algorithms, our

[1] We will define the *worst case error*, the *spline algorithm*, and all the other needed concepts later.

formulation does not require any regularization (see [8, 9] for a review of vision problems requiring regularization.)

Section 4 deals with the implementation of the optimal algorithm. Its performance is analyzed in terms of the error it creates in recovering a surface. We will show sample runs that achieve a good approximation with a small number of data.

In section 5 we discuss the cost of the proposed algorithm. We show how to take advantage of the structure of the data and modify the algorithm, so that significant cost improvements are obtained. Furthermore, the algorithm can now be implemented in parallel, resulting in an even further reduction in the running time.

2. Formulation of the problem.

In the introduction of the paper we said that our aim is to recover a 2-dimensional slice of a 3-dimensional surface. A 2-dimensional slice (see Fig. 1) can be seen as a function of one variable $f : \mathbb{R} \longrightarrow \mathbb{R}$ belonging in the space of functions F_0. Our aim is to obtain an approximation $x \in F_0$ to our function $f \in F_0$ using the data that we can derive from the shadows. We want the approximation x to be as close to f as possible.

2.1 Function Space.

Let,

$$F_0 = \left\{ f \mid f : [0,1] \longrightarrow \mathbb{R},\ f' \text{ absolutely cont.},\ \|f''\|_{L_2} \leq 1 \right\}, \tag{2-1}$$

be the space that contains the functions f that we want to approximate.[2] [3] The norm $\|\cdot\|_{L_2}$ is defined as $\|f\|_{L_2} = \sqrt{\int_0^1 |f(x)|^2 dx}$.

Also, define the bilinear form $\langle \cdot, \cdot \rangle$ to be such that,

$$\langle f, g \rangle = \int_0^1 f''(x)\, g''(x)\, dx, \tag{2-2}$$

and the norm $\|\cdot\|$ to be such that,

$$\|f\| = \langle f, f \rangle^{1/2}. \tag{2-3}$$

Clearly $\langle \cdot, \cdot \rangle$ defined above is a semi-inner product and $\|\cdot\|$ is a semi-norm. If we pose the additional requirements $f(0) = 0$ and $f'(0) = 0$ on our function, then $\langle \cdot, \cdot \rangle$ is an inner product and $\|\cdot\|$ a norm. Consequently, F_0 equipped with $\langle \cdot, \cdot \rangle$ is a semi-Hilbert space or a Hilbert space respectively.

2.2 Information.

In the next step we will extract from the image(s) the information, that is contained in the shadows, and which will be denoted by $N(f)$.[4] Clearly, from the position of the

[2] The bound of 1 in $\|f''\|_{L_2}$ is assumed without loss of generality. However, as we already mentioned, any fixed bound is equally good.

[3] The use of the interval $[0, 1]$ is not restrictive either. Any interval $[a, b]$ for some a and b is equally good.

[4] The concept of information is considerably different from the concept of data. The data vector is a vector of fixed values, while information is an operator. We will use the term somewhat imprecisely. The user is referred to [11, 12, 13, 14] for a more detailed discussion on the concept of information operators.

light source, we can immediately obtain the derivative of the function f at the point x_i, x_i being the beginning of the shadow (see Fig. 2). We can also obtain the difference between the two function values $f(x_i) - f(\bar{x}_i)$, at the beginning and at the end of the shadow respectively, given by $f'(x_i)(x_i - \bar{x}_i)$. Assuming that $l_i(x)$ is the straight line segment passing through the points x_i, $\bar{x}_i$, an additional piece of information can be obtained. It holds (see Fig. 2) that,

$$f(x) \;<\; l_i(x), \quad \forall x \in [x_i, \bar{x}_i]. \tag{2-4}$$

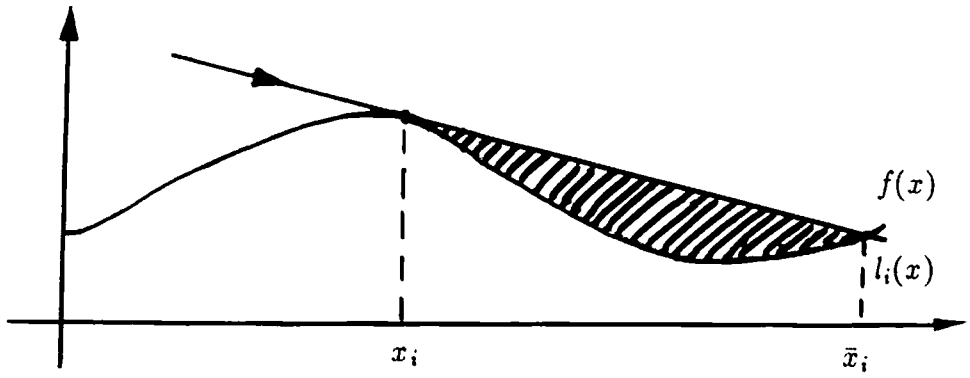

FIGURE 2.

So, formally, the information $N(f)$ contains triplets,

$$\big\langle f'(x_i), f(x_i) - f(\bar{x}_i), f(x) < l_i(x) \big\rangle. \tag{2-5}$$

Note that the third item in the triplet is a consistency condition.

In each one of the images in our sample there are 0, 1 or more shadowed areas. From each one of those shadowed areas we can obtain a triplet of the form (2-5). If we group all the data resulting from this sampling we obtain the vector,

$$\begin{aligned} N(f) = \big[& f'(x_1), \ldots, f'(x_n), f(x_1) - f(\bar{x}_1), \ldots, \\ & f(x_n) - f(\bar{x}_n), f(x_1) - f(t_1), \ldots, f(x_1) - f(t_{m_1}), \ldots, \\ & f(x_2) - f(t_{m_2}), \ldots, f(x_n) - f(t_{m_n}) \big]^{\mathsf{T}}, \end{aligned} \tag{2-6}$$

where $m_1, \ldots, m_n$ are the number of points t_i in every interval $[x_i, \bar{x}_i]$ for which (2-4) holds, and $m_1 + \cdots + m_n = m$. Clearly, while we know that there are exactly $2n$ pieces of information in the first part of $N(f)$, we cannot bound the cardinality of the last part because, it can be the case that $m \longrightarrow +\infty$.

3. Solution of the problem - The optimal algorithm.

We now proceed to the solution of the problem. We want, given information $N(f)$, to obtain an algorithm φ that will provide an approximation to our function f. An *algorithm* is defined as any mapping from the space of all permissible data vectors to the space F_0.

3.1 Algorithm error.

The error of an algorithm for a given fixed function f is given by $\|f - \varphi(N(f))\|$. We would like to know what is the largest possible error that can be made by the algorithm, i.e. we want the error of the algorithm for the worst possible function f.

DEFINITION 3.1. *The worst case error of an algorithm φ is,*

$$e(\varphi, N(f)) = \sup_{\bar{f} \in F_0} \{\|\bar{f} - \varphi(N(\bar{f}))\|, \ N(\bar{f}) = N(f)\}. \tag{3-1}$$

Clearly, we want an algorithm that minimizes $e(\varphi, N(f))$.

DEFINITION 3.2. *An algorithm φ^* that has the property,*

$$e(\varphi^*, N(f)) = \inf_{\varphi} \{e(\varphi, N(f))\}, \qquad \forall f \in F_0 \tag{3-2}$$

is called a strongly optimal error algorithm.

The quantity at the right side of (3-2), i.e. the infimum of the error of all algorithms solving the problem given information $N(f)$, is a property of the problem itself, and does not depend on the particular algorithm used at any moment. This quantity, gives the inherent uncertainty of the problem for given information, and is called the ***radius of information***. Clearly, the error of the strongly optimal algorithm equals the radius of information.[5]

3.2 The spline algorithm

We propose the spline algorithm φ^s for the solution of our problem. Splines have been known to give the optimal solution to many interesting problems [1, 2, 6, 7, 11, 12].

DEFINITION 3.3. *A spline σ is an element in the space of functions F_0 such that,*

(1) $N(\sigma) = \vec{y}$.
(2) $\|\sigma\| = \min_{f \in F_0} \{\|f\|, \quad N(f) = \vec{y}\}$.

The meaning of (1) is that the spline must interpolate the data, and (2) says that the spline is the function that minimizes $\|\cdot\|$. The spline algorithm is the process that constructs the spline.

In a Hilbert space setting one can obtain a closed form for the spline algorithm. In our particular case, the spline algorithm is given by,

$$\varphi^s(x) = \sum_{i=1}^{2n} a_i g_i(x) + \sum_{j=1}^{m} c_j h_j(x), \tag{3-3}$$

where $\{g_i\}_{i=1,\ldots,2n}$ and $\{h_j\}_{j=1,\ldots,m}$ are such that,

$$g_i''(x) = \frac{(x_i - x)_+^0 - (x_{i-1} - x)_+^0}{\sqrt{x_i - x_{i-1}}}, \qquad i = 1, \ldots, n \tag{3-4}$$

where $(x_i - x)_+^0 = 1$ for $x_i > x$ and 0 otherwise,

$$g_{n+i}''(x) = (\bar{x}_i - x)_+ - (x_i - x)_+ - (x_i - x)_+^0 (\bar{x}_i - x_i), \qquad i = 1, \ldots, n \tag{3-5}$$

[5] One point that must be mentioned here is that the radius of information describes, as we said before, the *inherent uncertainty of the problem* and has a specific value, say R. However small or large R may be, there is no procedure that will guarantee error less than that.

where $(x_i - x)_+ = x_i - x$ for $x_i > x$ and 0 otherwise, and

$$h_j''(x) = (t_j - x)_+ - (x_i - x)_+ - (x_i - x)_+^0(t_j - x_i), \quad i = 1, \ldots, n, \ j = 1, \ldots, m. \quad (3\text{-}6)$$

The functions $\{g_i\}_{i=1,\ldots,2n}$ and $\{h_j\}_{j=1,\ldots,m}$ are the *representers* of the functionals that construct the information $N(f)$, properly modified to have a small area of support.

The coefficients a_i and c_j are chosen so that the definition of the spline is satisfied, that is, φ^s interpolates the data, and also minimizes the norm $\|\cdot\|$. If the area of support of every g_i is disjoint from the area of support of every other and furthermore $\langle g_i, g_j \rangle = \delta_{ij}$, the Krönecker delta, then the coefficients a_i are given directly by the theory. This is not the case in this setting where the coefficients a_i are obtained by solving a system of linear equations and the coefficients c_j are obtained by directly minimizing $\|\cdot\|$. We will define the minimization problem in section 3.3 and describe the implementation of the algorithm in section 4.

For the spline algorithm the following very strong theorem holds [7, 12].

THEOREM 3.1. *Let F_0 be a Hilbert space, $f \in F_0$ and information $\vec{y} = N(f)$. Then, the spline algorithm interpolating the data $\vec{y}$ exists, is unique, and achieves error at most twice the radius of information.*

From Theorem 3.1 we can obtain two very important results. First, our problem under the proposed formulation is well-posed. This property [3, 10] is always desirable when solving a problem. Computer vision problems tend to be ill-posed and considerable effort has been spent by the vision community towards the correct formulation that will yield well-posedness (See [4, 8, 9] for a survey.)

Second, the spline algorithm has a worst case error that is within a factor of 2 from the radius of information. The algorithm that achieves that is called *almost strongly optimal* [11]. If the problem is *linear*[6] then the spline algorithm φ^s has a worst case error equal to the radius of information and is, therefore, the strongly optimal algorithm.

The shape from shadows problem is linear, only if the cardinality of the second part of the information $N(f)$ is 0, i.e. $m = 0$. If $m > 0$ then φ^s is not strongly optimal, but almost strongly optimal as derived from Theorem 3.1.[7]

3.3 The minimization problem.

We want to minimize $\|\sigma\|^2$ where σ is given by (3-3). We can write,

$$\begin{aligned}
\|\sigma\|^2 &= \langle \sigma, \sigma \rangle \\
&= \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} a_{i_1} a_{i_2} \langle g_{i_1}, g_{i_2} \rangle + 2 \sum_{i=1}^{n} \sum_{j=1}^{k} a_i c_j \langle g_i, h_j \rangle + \sum_{j_1=1}^{k} \sum_{j_2=1}^{k} c_{j_1} c_{j_2} \langle h_{j_1}, h_{j_2} \rangle \\
&= \vec{a}^{\,\mathsf{T}} \mathbf{G}\, \vec{a} + 2\, \vec{a}^{\,\mathsf{T}} \mathbf{P}^{\mathsf{T}} \vec{c} + \vec{c}^{\,\mathsf{T}} \mathbf{H}\, \vec{c}, \qquad (3\text{-}7)
\end{aligned}$$

where $\mathbf{G} = \{\langle g_i, g_j \rangle\}_{i,j=1,\ldots,2n}$, $\mathbf{P} = \{\langle h_j, g_i \rangle\}_{\substack{j=1,\ldots,m \\ i=1,\ldots,2n}}$, and $\mathbf{H} = \{\langle h_i, h_j \rangle\}_{i,j=1,\ldots,m}$.

[6] For an exact definition of a linear problem see [7, 12, 13, 14].

[7] Strongly optimal algorithms for non-linear problems are not known in general, and if they are, they can be very difficult and expensive to calculate.

Also,

$$\begin{aligned} \langle \sigma, g_s \rangle &= \sum_{i=1}^{n} a_i \langle g_i, g_s \rangle + \sum_{j=1}^{k} c_j \langle h_j, g_s \rangle = y_s \\ \Longrightarrow \quad & \mathbf{G}\,\vec{a} + \mathbf{P}\,\vec{c} = \vec{y} \\ \Longrightarrow \quad & \vec{a} = \mathbf{G}^{-1}\left(\vec{y} - \mathbf{P}\,\vec{c}\right), \end{aligned} \tag{3-8}$$

and,

$$\begin{aligned} \langle \sigma, h_s \rangle &= \sum_{i=1}^{n} a_i \langle g_i, h_s \rangle + \sum_{j=1}^{k} c_j \langle h_j, h_s \rangle \leq A_s \\ \Longrightarrow \quad & \mathbf{P}^{\top}\vec{a} + \mathbf{H}\,\vec{c} \;\leq\; \vec{A} \\ \overset{(3\text{-}8)}{\Longrightarrow} \quad & \mathbf{P}^{\top}\mathbf{G}^{-1}\vec{y} - \mathbf{P}^{\top}\mathbf{G}^{-1}\mathbf{P}\,\vec{c} + \mathbf{H}\,\vec{c} \;\leq\; \vec{A} \\ \Longrightarrow \quad & \left(\mathbf{H} - \mathbf{P}^{\top}\mathbf{G}^{-1}\mathbf{P}\right)\vec{c} \;\leq\; \vec{A} - \mathbf{P}^{\top}\mathbf{G}^{-1}\vec{y}. \end{aligned} \tag{3-9}$$

Now, if we substitute (3-8) for $\vec{a}$ in (3-7) we obtain,

$$\begin{aligned} \|\sigma\|^2 &= \vec{c}^{\top}\mathbf{H}\,\vec{c} + 2\left(\mathbf{G}^{-1}\left(\vec{y} - \mathbf{P}\vec{c}\right)\right)^{\top}\mathbf{P}\,\vec{c} + \left(\mathbf{G}^{-1}\left(\vec{y} - \mathbf{P}\vec{c}\right)\right)^{\top}\mathbf{G}\,\left(\mathbf{G}^{-1}\left(\vec{y} - \mathbf{P}\vec{c}\right)\right) \\ &= \qquad\qquad \cdots \\ &= \vec{c}^{\top}\left(\mathbf{H} - \mathbf{P}^{\top}\mathbf{G}^{-1}\mathbf{P}\right)\vec{c} + \vec{y}^{\top}\mathbf{G}^{-1}\vec{y}. \end{aligned} \tag{3-10}$$

Since $\vec{y}^{\top}\mathbf{G}^{-1}\vec{y}$ has a known fixed value for any given problem, it remains to minimize $\vec{c}^{\top}\left(\mathbf{H} - \mathbf{P}^{\top}\mathbf{G}^{-1}\mathbf{P}\right)\vec{c}$ given the conditions in (3-9). This is a quadratic minimization problem that can be solved using a standard method.

The cardinality of the non-linear part of the information is not fixed and the choice of the value of m must be performed carefully, especially since the minimization process is costly. One may choose to always ignore the non-linear information encoded in (2-4). On the other hand, non-linear information of some fixed cardinality may be always included, regardless of whether the constraints in (2-4) are violated or not.

We choose an intermediate approach which will always assure that (2-4) holds, and at the same time will minimize the cost of the algorithm (see sections 4 and 5.)

4. Application of the algorithm - Numerical runs.

The spline algorithm of section 3 has been applied and its performance has been tested in practice.

4.1 Algorithm implementation.

In our implementation the calculation of the spline algorithm proceeds in steps. From our early experience with experimental systems we have concluded that except very few cases, the non-linear part of the information is not needed. This means that the approximation produced by $\varphi^s(x)$ using information $N(f)$ with $m = 0$ does not violate the constraints (2-4).

Stage 1:

Therefore, we begin the implementation of the spline algorithm by assuming that $m = 0$ and we will first construct the values of the coefficients a_i. This is done by solving the system of equations,

$$\mathbf{G}\,\vec{a} = \vec{y}, \tag{4-1}$$

where $\mathbf{G} = \{\langle g_i, g_j \rangle\}_{i,j=1}^{2n}$ and $\{g_i\}_{i=1,\ldots,2n}$ are given by (3-4) and (3-5). The system is solved by a direct method without the need for pivoting since it is symmetric, positive definite and has a nice structure that reduces the number of calculations.

As a next step, we use the computed values of the a_i' s to construct the spline algorithm, and we plot its graph.

Third, we check to see whether the non-linear constraints are violated. This is done on line while we are plotting $\varphi^s(x)$.

If the non-linear constraints (2-4) are not violated, which as we mentioned before is usually the case, we do not need to do anything else. We have already obtained the approximation $\varphi^s(x)$ to the function f and we have plotted it. Also, since we have not used the non-linear part of the information, the problem is linear hence the spline algorithm achieves the radius of information.

Stage 2:

If the constraints (2-4) are violated, then we do not have a sufficiently good approximation, which means that we must obtain the coefficients c_j of (3-3). To do so, we have to solve the minimization problem derived in section 3.3.

We will consequently proceed as follows. We will take a few points t_i in the shadowed intervals where the constraints are violated. For these points we solve the minimization problem. Then we check again for violations of the non-linear constraints. If there are violations we repeat Stage 2. We select a few more points from the interval(s) where (2-4) is violated, and we add them to the sample. The minimization is repeated for the new set of points and the new coefficients are derived. At the same time, the a_i' s and the old c_j' s are modified.

We perform the minimization for a few points at a time for various reasons.

(1) It is a costly process and we would like to keep the dimensions of the problem as small as possible.
(2) At each new iteration we do not need to undo our previous work, but we simply modify the existing coefficients while deriving the new ones.
(3) We rarely need to use more that one or two points per shadowed area.

4.2 Test runs.

We have constructed a broad series of functions and we have run our algorithm on them. We started from the smoothest possible function which is a trigonometric one. Fig. 3 shows the graph of this function, and the graph of the approximating spline. A broken line is used to draw the function and a solid one is used to draw the approximating spline.

The information we have used has been obtained for only 2 different light positions, and already yields a very close approximation. If we add samples from another 2 light positions

for a total of 4, the approximation is so close to the function f that the discrepancy cannot be observed.

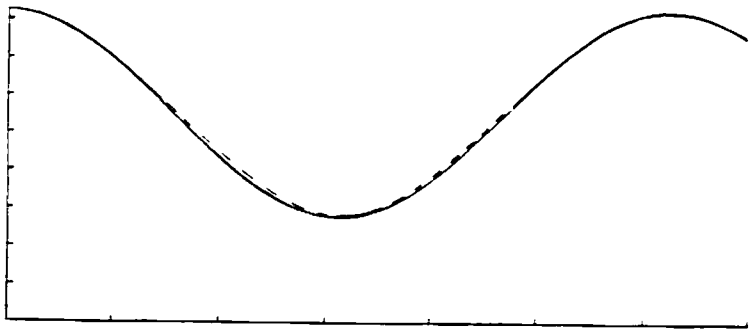

FIGURE 3.

From the quality of this approximation we may assume that smooth functions can be approximated very well. We will move now to the other side of the spectrum which contains functions with as few derivatives as possible. This proves actually to be the most difficult case. In our setting the most irregular functions are the ones that have continuous first derivatives and discontinuous second ones. Piecewise quadratic polynomials with different second derivative from piece to piece, are functions of this type. We built many of these functions with as many as 40 different pieces each.

Additionally, in order to magnify the visual effect of any discrepancy between the function and the approximation we relaxed the assumption $\|f''\| \leq 1$ that we posed in our space definition. Otherwise, the difference between f and φ^s would not be visible in any test run we would choose to show.

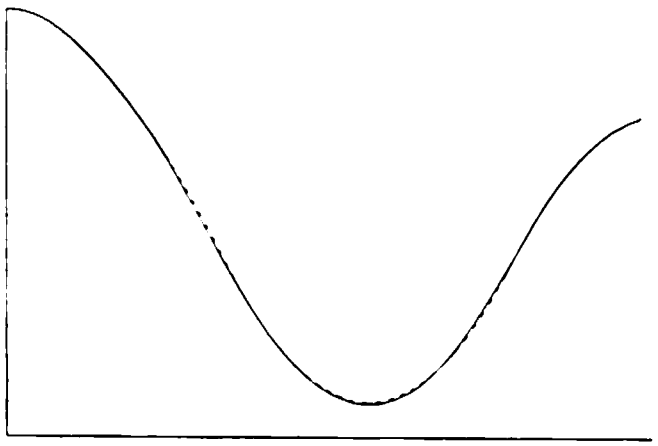

FIGURE 4.

In Figure 4 we can see the approximation to a function consisting of 10 piecewise polynomials of degree 2. The information used has been obtained from 6 different lighting angles.

It can again be seen that the function f and the approximating spline φ^s almost coincide.

We will now show one of the most difficult functions we have built, together with its approximating spline. The function consists of 40 piecewise polynomials of second degree, and has very large jumps in its second derivative, hence it has large $\|f''\|$. The information we have used to compute $\varphi^s(x)$, shown in Figure 5, is derived by using 8 different light angles.

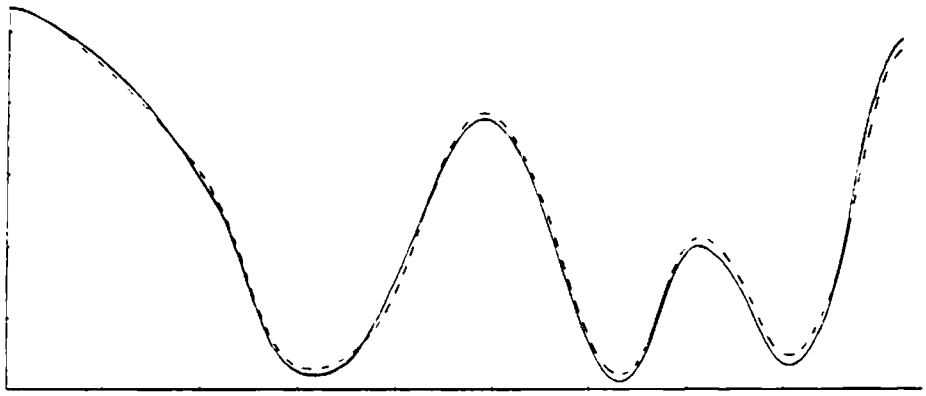

FIGURE 5.

Another issue needs to be discussed. Namely, in all the above cases the approximation to the given function has been constructed without the use of the non-linear information which, as we have mentioned, is usually the case. We will contrive a case where the use of the constraints (2-4) is needed, so that we can exhibit the second stage of the algorithm (3-3).

When,

(1) $f''(x)$ varies a lot from one polynomial piece to the other,
(2) The sampling is sparse,
(3) Both light positions are from the same side of the horizon.

Then, the approximation given by the first stage of the algorithm can fail to satisfy (2-4) (See Fig. 6.)

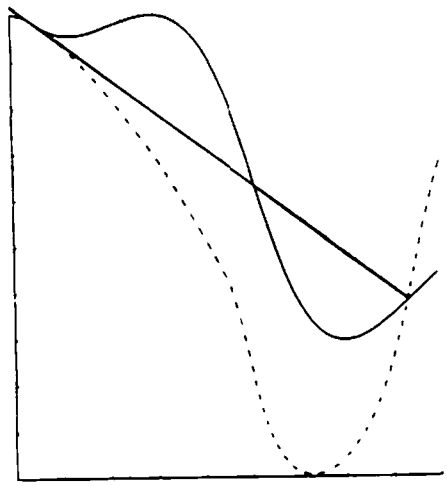

FIGURE 6.

In the example of Fig. 6 only 2 light angles were used, both from the same side of the horizon, and far apart from each other. We therefore used the second stage of the algorithm,

we added two extra points t_1 and t_2 and we included $f(t_1) < l_1(t_1)$, $f(t_2) < l_1(t_2)$ in $N(f)$. Then, the constructed approximation (Fig. 7) satisfies (2-4).

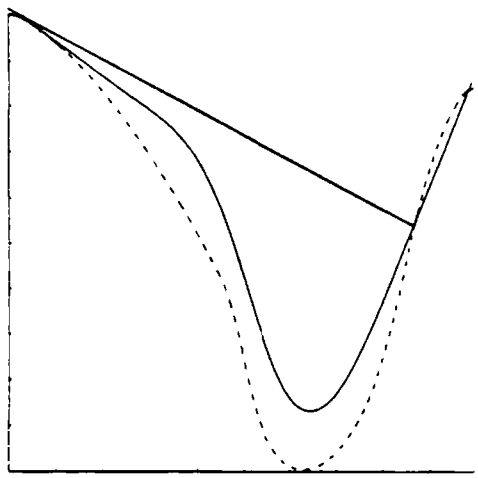

FIGURE 7.

5. Cost of the algorithm - Speed improvements.

5.1 Algorithm cost.

Let us now discuss the speed performance of our algorithm. The spline algorithm, as defined in section 3, is linear in terms of its input. Thus, if we knew the coefficients a_i and c_j then, $cost(\varphi^s)$ would be $O(n)$.

In our case, the coefficients of the spline algorithm are not known, and must be constructed. To achieve this we must solve a system of linear equations, and sometimes, a minimization problem. These costs dominate the cost of the algorithm.

In particular the solution of the system (4-1) has a cost $O(n^3)$. The cost of the quadratic minimization is considerably higher, that is it is exponential in terms of the number of nonlinear information samples used in each stage. In this case, and since we have observed that better and denser sampling alleviates the need to use the second stage of the algorithm, we might choose to increase the cardinality of the sampling, if that can be done, and solve a slightly larger linear problem instead.

5.2 Speed improvements.

In section 5.1 we have discussed the cost of a very straightforward implementation of the spline algorithm described in section 4.1. We now show that a slight improvement in the implementation of the algorithm can yield a significant speedup. This speedup can be achieved only if the function we want to recover can be split in distinct sections that we will from now on call *valleys*. A valley is defined by two local maxima of the function, but also depends on the specific sampling. For example, the function of Fig. 1 has 2 valleys. In particular, we say that the function f, under some fixed sampling, has k valleys if we can define k partitions $\Pi_1, \Pi_2, \ldots, \Pi_k$ of the functions $\{g_i\}_{i=1,\ldots,2n}$, given by (3-4) and (3-5), such that the union of the areas of support of all the functions in each partition is disjoint from the union of the areas of support of the functions in every other partition.

We can detect the existence of any number of valleys in time $O(n)$ and subsequently, we can solve k problems of sizes $n_1, n_2, \ldots, n_k$ respectively, instead of solving one problem of size n, where $n = n_1 + n_2 + \cdots + n_k$.

To connect the pieces resulting from each of the k problems we need constant time per problem, hence combining can be done in time $O(k)$.

Therefore, the total cost of this algorithm, which we will denote φ_i^s, will be $O(k\nu^3)$, where $\nu = \max\{n_1, \ldots, n_k\}$.

5.3 Parallel implementation.

Since splitting the problem into individual subproblems and combining the resulting surfaces is straightforward and cheap to implement, one immediate extension to the above set-up of the problem is to assign one individual subproblem to a different processor and solve the initial problem in a parallel or in a distributed environment.

Again, splitting into k subproblems requires time $O(n)$ and combining the individual solutions into one requires time of $O(k)$. Then every processor will require time $O(n_i^3), \quad i = 1, \ldots, k$ resulting in a total cost for the parallel version φ_p^s of our algorithm of $O(\nu^3)$, where $\nu = \max\{n_1, \ldots, n_k\}$.

Let us now compare the three different implementations of the spline algorithm, using a specific example. Assume we have information $N(f)$ of cardinality $n = 512$.[8] Also assume that the number of valleys k is 8. We assume without loss of generality that $n_1 = n_2 = \cdots = n_k = \nu = 64$. For these values of n, ν, and k the performance figures listed in Table 1 are obtained.

Algorithm	Cost (Millions of ops)
φ^s	$512^3 \approx 134.4$
φ_i^s	$8 \cdot 64^3 \approx 2.0$
φ_p^s	$64^3 \approx 0.26$

TABLE 1.

It is apparent from the above table that we can obtain substantial speed improvements with very low added overhead. It should additionally be noted that we can obtain a good approximation using around 8 to 10 different light positions. If this is the case, the only occasion when we can obtain $N(f)$ of cardinality $n = 512$ is if k is very large, i.e. if we have a big number of valleys. In that case the speed improvements should be even larger than the ones already exhibited. Conversely, if the number of valleys is small then the size of n is expected to be low, since it is proportional to the number of light positions, and will in no case reach the magnitude of the above example.

6. Conclusion - Future work.

We solved the problem of recovering a one-dimensional surface slice from the shadows it casts on itself when lighted by a light source positioned at various locations.

[8] A very high number compared to the values of n used in the sample runs we have shown.

We proposed a formulation that results in a well-posed problem and we have consequently proceeded into solving it. We proposed an optimal error algorithm which additionally achieves a low time cost, especially if a clever but simple breakdown of the problem is used.

There are many aspects of this problem that can be looked at in the future. A method for the faster solution of the optimization problem, based on its specific structure, is one of them. On the other hand it would be useful to quantify whether the use of the non-linear information can be avoided. Our current belief is that good sampling can take care of all cases. The second stage of the algorithm might be still useful in cases where we have to deal with a fixed, given, not very appropriate sampling.

Another natural extension to the shape from shadows problem is to try to recover the whole 3-D surface instead of recovering surface slices, as we are doing in this paper. We are currently working on this interesting aspect.

7. Acknowledgements.

I would like to thank Prof. Greg Wasilkowski, currently with the University of Kentucky at Lexington, without whose guidance this work would not have been possible.

References

[1] Anselone, P. M., and Laurent, P. J., *A general method for the construction of interpolating or smoothing spline functions*, Nummer. Math. 12 (1968).

[2] Atteia, M., *Fonctions spline généralisées*, C.R. Acad. Sci. Paris 261 (1965).

[3] Hadamard, J., *Sur les problèmes aux derivées partielles et leur signification physique*, Princeton Univ. Bulletin 13 (1902).

[4] Hatzitheodorou, M. G., *The Application of Approximation theory methods to the solution of computer vision problems*, (In Progress).

[5] Kender, J. R., and Smith, E. M., *Shape from darkness. Deriving surface information from dynamic shadows*, Proceedings AAAI (1986).

[6] Holmes, R., *R-splines in Banach spaces : I. Interpolation of linear manifolds*, J. Math. Anal. Appl. 40 (1972).

[7] Michelli, C. A., and Rivlin, T. J, *A Survey of optimal recovery*, in "Optimal estimation in Approximation theory," Plenum Press, 1977.

[8] Poggio, T., *Computer vision*, MIT Artificial Inteligence Lab (1986).

[9] Poggio, T., and Torre, V., *Ill-posed problems and regularization analysis in early vision*, MIT Artificial Inteligence Lab (1984).

[10] Tikhonov, A. N., and Arsenin, V. Y., "Solutions of ill-posed problems," V.H.Winston and Sons, 1977.

[11] Traub, J. F., Wasilkowski, G., and Woźniakowski, H., "Information, Uncertainty, Complexity," Addison-Wesley, 1983.

[12] Traub, J. F., and Woźniakowski, H., "A general theory of optimal algorithms," Academic Press, 1981.

[13] Woźniakowski, H., *A survey of information-based complexity*, Journal of Complexity 1 (1985).

[14] Woźniakowski, H., *Information-based complexity*, Annual Review of Computer Science 1 (1986).