Quadratic Shading and its Hardware Implementation

Abbas Ali Mohamed, László Szirmay-Kalos, Tamás Horváth and Tibor Fóris
Department of Control Engineering and Information Technology,
Technical University of Budapest
Budapest, Pázmány P. 1/D, H-1117, HUNGARY
(abbas,szirmay)@seeger.iit.bme.hu

Abstract. Rendering systems often represent curved surfaces as a mesh of planar polygons that are shaded to restore a smooth appearance. Gouraud shading uses linear color interpolation and its hardware implementation is relatively easy, but it handles specular highlights incorrectly and introduces annoying artifacts called Mach banding over the edges of the polygon mesh. In software rendering Phong shading has been more popular, because it can realistically handle specular materials. Since it requires the rendering equation to be evaluated for each pixel, its hardware support poses problems. This paper presents a nonlinear, i.e. quadratic interpolation scheme which is in between Gouraud shading and Phong shading. It can also be implemented in hardware as Gouraud shading but its shading quality is comparable with that of the Phong shading. The software simulation and the VHDL description of the shading hardware are also presented.

Keywords: Reflectance function, BRDF representation, hardware rendering, incremental concept, interpolation, Gouraud and Phong shading.

1. Introduction

Computer graphics aims at rendering complex virtual world models and presenting the image for the user. To obtain an image of a virtual world, surfaces visible in pixels are determined, and the rendering equation or its simplified form is used to calculate the intensity of these surfaces, defining the color values of the pixels. The rendering equation [9] expresses the output radiance $I^{out}(\vec{x}, \vec{V})$ of a surface point $\vec{x}$ at direction $\vec{V}$ as the function of the local surface properties and the incoming radiance I^{in} emitted by the light sources or reflected off other surfaces from direction $\vec{L}$:

$$I^{out}(\vec{x}, \vec{V}) = \int_{\Omega'} I^{in}(\vec{x}, \vec{L}) \cdot f_r(\vec{L}, \vec{x}, \vec{V}) \cdot \cos\theta^{in} d\omega_{\vec{L}} \tag{1}$$

where θ^{in} is the angle between the incoming direction $\vec{L}$ and the surface normal at the reflection point $\vec{x}$, i.e. $\cos\theta^{in} = \vec{L} \cdot \vec{N}$ if the vectors have unit length, f_r is the BRDF (bi-directional reflected distributed function), and Ω' is the set of possible incoming directions forming a hemisphere.

If the indirect illumination coming from other surfaces is ignored and only directional and positional light sources are present, I^{in} is a Dirac-delta type function which simplifies

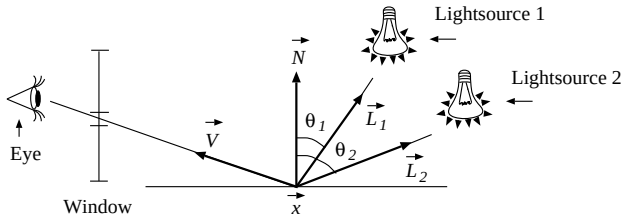

Fig. 1. Radiance calculation in local illumination methods

the integral to a discrete sum:

$$I^{out}(\vec{x},\vec{V}) = \sum_l I_l^{in}(\vec{x},\vec{L}_l) \cdot f_r(\vec{L}_l,\vec{x},\vec{V}) \cdot \cos\theta_l^{in} \tag{2}$$

where I_l^{in} is the incoming radiance generated by light source l (figure 1). Note that this model does not account for the multiple reflections of the light, only the direct illumination of the light sources is considered.

The BRDF function f_r is responsible for the optical properties of the surface. In practice BRDF functions are mathematical formulae that have some free parameters that can be set to mimic a given material.

2. Gouraud and Phong shading

The radiance values are needed for each pixel, which, in turn, require the rendering to be solved for the visible surface. The rendering equation, even in its simplified form, contains a lot of complex operations, including the computation of the vectors, their normalization and the evaluation of the output radiance, which makes the process rather resource demanding.

The speed of rendering could be significantly increased if it were possible to carry out the expensive computations just for a few points or pixels, and the rest could be approximated from these representative points by much simpler expressions. One way of obtaining this is the tessellation of the original surfaces to polygon meshes and using the vertices of the polygons as representative points. In this paper only triangle mesh models are considered, thus the geometry should be approximated by a triangle mesh before the algorithms can be used. It is assumed that the geometry has been transformed to the screen coordinate system suitable for visibility calculations and projection. In the screen coordinate system the X, Y coordinates of a point are equal to the corresponding coordinates of that pixel in which this point can be seen, and the Z coordinate increases with the distance from the viewer, thus it is the basis of visibility calculations (figure 2). Note, on the other hand, that the vectors used by the rendering equation are not trans-

formed, because the viewing transformation is not angle preserving thus it may distort the angles between them.

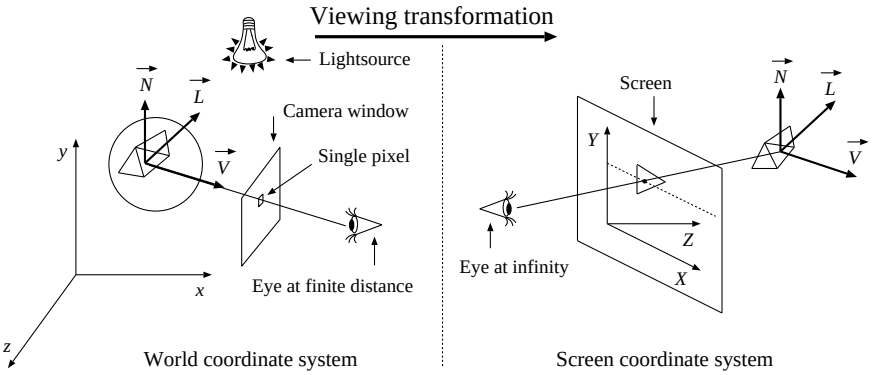

Fig. 2. Transformation to the screen coordinate system

As mentioned, interpolation can be used to speed up the rendering of the triangle mesh, where the expensive computations take place just at the vertices and the data of the internal points are interpolated. A simple interpolation scheme would compute the color and linearly interpolate it inside the triangle (Gouraud shading [1]). However, specular reflections may introduce strong non-linearity, thus linear interpolation can introduce severe artifacts (left of figure 3). The core of the problem is that the color can be a strongly non-linear function of the pixel coordinates, especially if specular highlights occur on the triangle, and this non-linear function can hardly be well approximated by a linear function.

The artifacts of Gouraud shading can be eliminated by a non-linear interpolation called Phong shading [2] (right of figure 3). In Phong shading, vectors used by the rendering equation are interpolated from the real vectors at the vertices of the approximating triangle. In simpler algorithms only the normal vectors are interpolated while the light and view vectors are constant. In more precise computations, the view and light vectors are also interpolated. The interpolated vectors are normalized and the rendering equation is evaluated at each pixel for diffuse and specular reflections and for each light source, which is rather time consuming. The main problem of Phong shading is that it requires complex operations on the pixel level, thus its hardware implementation is not feasible. Originally, the interpolating function is linear. For example, the normal vector of a pixel (X, Y) is

$$\vec{N}(X,Y) = a_1(X,Y) \cdot \vec{N}_1 + a_2(X,Y) \cdot \vec{N}_2 + a_3(X,Y) \cdot \vec{N}_3 \qquad (3)$$

where $a_i(X,Y) = a_{ix}X + a_{iy}Y + a_{i0}$ $(i = 1,2,3)$ is a linear weighting function and $\vec{N}_i$ is the normal vector at vertex i. The interpolation criterion requires that $a_i(X,Y) = 1$ at

vertex i and 0 in the other two vertices. From this criterion, the parameters (a_{ix}, a_{iy}, a_{i0}) of each weighting function can be determined.

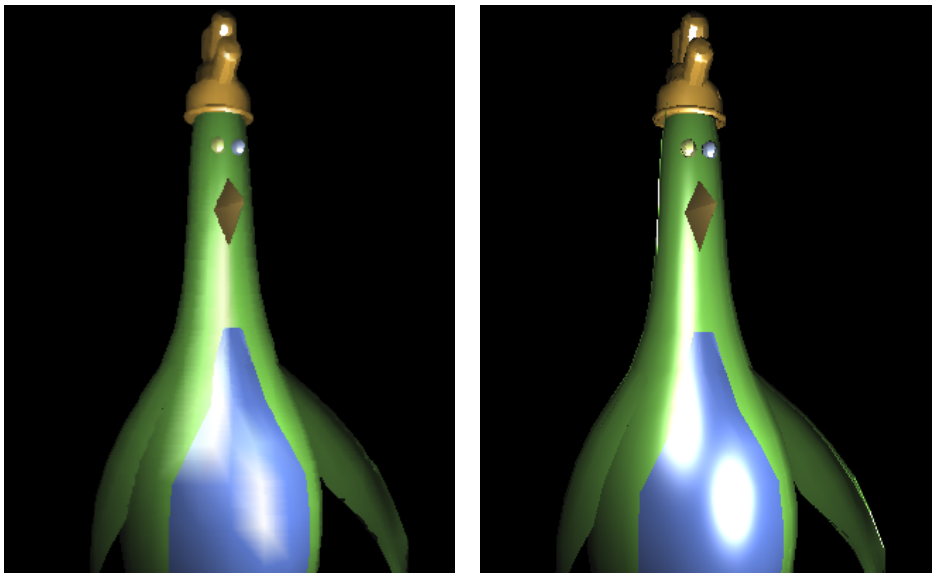

Fig. 3. Comparison of linear interpolation i.e. Gouraud shading (left) and non-linear interpolation by Phong shading (right).

The superior rendering quality of Phong shading forced research to try to find a reasonable compromise between Gouraud and Phong algorithms, that keeps the image quality but also allows for hardware implementation. In Textronix terminals, for example, the method called pseudo-Phong shading was implemented. Pseudo-Phong shading recursively decomposes the triangles into small triangles setting the vectors at the vertices according to a linear formula, and uses Gouraud shading when the small triangles are rendered. If the sizes of the small triangles are comparable to the size of the pixels, then this corresponds to Phong shading. However, when they are close to the original triangle, this corresponds to Gouraud shading. Another family of algorithms used highlight tests [6] to determine whether or not a specular highlight intersects the triangle. If there is no intersection, then Gouraud shading is used, otherwise the triangle is rendered with Phong shading. Duff [3] extended the incremental approach of Gouraud shading to Phong shading. He obtained the reflected radiance of a single light source as a direct function of the pixel coordinates and evaluated this function using forward differences. Based on this, Bishop proposed a simplification using Taylor's approximation in [4]. The determination of the derivatives of the reflected radiance is quite complicated

and requires expensive computation, and this computation must be repeated at each pixel for diffuse and specular reflections and for each light source. Besides, according to the nature of Taylor's series, the approximation is good around the point where the derivatives were computed. Neighboring triangles may have different color variation on their edges, which leads to Mach banding over the edges of the triangles. Claussen [7] compared different simplification strategies of the Phong illumination formulae and vector interpolation. Spherical interpolation elegantly traces back the interpolation to the interpolation of a single angle inside a scan-line [5]. However, finding the parameters of a scan-line is also rather complicated and the method requires the evaluation of the rendering equation at each pixel and for each light source. The computational cost is also proportional to the number of light sources.

In this paper we propose a new interpolation scheme that uses appropriately selected quadratic functions which can be implemented in hardware and can be initialized without the computational burden of the Taylor's series approach. Unlike previous techniques the new method can simultaneously handle arbitrary number of light sources and arbitrary BRDF models.

3. Quadratic color interpolation

Our approach is in between Gouraud shading and Phong shading. The rendering equation is evaluated in a few representative points and the interpolation is done in color space as in Gouraud shading. However, the interpolation is not linear, but rather quadratic. Since the quadratic function has six degrees of freedom, the rendering equation will be evaluated at six representative points on the triangle and the color is interpolated from the colors of these representative points. Let us approximate the color inside the triangle by the following two-variate quadratic function:

$$I(X,Y) = T_5X^2 + T_4XY + T_3Y^2 + T_2X + T_1Y + T_0. \tag{4}$$

To find the unknown parameters $T_0, \ldots, T_5$, the color obtained from the rendering equation is substituted into this scheme at six points, and the six variate linear equation is solved for the parameters. The selection of these representative points should take into account different criteria. The error should be roughly uniform inside the triangle but should be less on the edges and on the vertices in order to avoid Mach banding. On the other hand, the resulting linear equation should be easy to solve in order to save computation time. An appropriate selection meeting both requirements uses the three vertices:

$$I(X_1,Y_1) = I_1, \quad I(X_2,Y_2) = I_2, \quad I(X_3,Y_3) = I_3,$$

and other three points on the edges half way between the two vertices, as follows:

$$I(\frac{X_1+X_2}{2}, \frac{Y_1+Y_2}{2}) = I_{12}, \quad I(\frac{X_1+X_3}{2}, \frac{Y_1+Y_3}{2}) = I_{13}, \quad I(\frac{X_2+X_3}{2}, \frac{Y_2+Y_3}{2}) = I_{23}.$$

Translating the triangle to have its bottom vertex at the coordinate origin yields:

$$\begin{aligned}
I_1 &= T_0,\\
I_2 &= T_5X_2^2 + T_4X_2Y_2 + T_3Y_2^2 + T_2X_2 + T_1Y_2 + T_0,\\
I_{12} &= T_5\frac{X_2^2}{4} + T_4\frac{X_2Y_2}{4} + T_3\frac{Y_2^2}{4} + T_2\frac{X_2}{2} + T_1\frac{Y_2}{2} + T_0,\\
I_3 &= T_5X_3^2 + T_4X_3Y_3 + T_3Y_3^2 + T_2X_3 + T_1Y_3 + T_0,\\
I_{13} &= T_5\frac{X_3^2}{4} + T_4\frac{X_3Y_3}{4} + T_3\frac{Y_3^2}{4} + T_2\frac{X_3}{2} + T_1\frac{Y_3}{2} + T_0,\\
I_{23} &= T_5\frac{(X_2+X_3)^2}{4} + T_4\frac{X_2+X_3}{2}\cdot\frac{Y_2+Y_3}{2}\\
&+ T_3\frac{(Y_2+Y_3)^2}{4} + T_2\frac{X_2+X_3}{2} + T_1\frac{Y_2+Y_3}{2} + T_0.
\end{aligned}$$

This system of linear equations can be solved in a straightforward way resulting in:

$$\begin{aligned}
T_0 &= I_1,\\
T_1 &= \frac{C_3X_2 - C_2X_3}{X_2Y_3 - Y_2X_3},\\
T_2 &= \frac{C_2Y_3 - C_3Y_2}{X_2Y_3 - Y_2X_3},\\
T_3 &= \frac{2C_{12} - T_5X_2^2 - T_4X_2Y_2}{Y_2^2},\\
T_4 &= \frac{(4C_{13}Y_2 - C_{23}Y_3)(2X_2^2Y_3 - 2X_2Y_2X_3) - (4C_{12}Y_3 - C_{23}Y_2)(2Y_2X_3^2 - 2X_2X_3Y_3)}{(2X_2^2Y_3 - 2X_2Y_2X_3)(Y_2X_3Y_3 - X_2Y_3^2) - (X_2Y_2Y_3 - Y_2^2X_3)(2Y_2X_3^2 - 2X_2X_3Y_3)},\\
T_5 &= \frac{(4C_{12}Y_3 - C_{23}Y_2)(Y_2X_3Y_3 - X_2Y_3^2) - (4C_{13}Y_2 - C_{23}Y_3)(X_2Y_2Y_3 - Y_2^2X_3)}{(2X_2^2Y_3 - 2X_2Y_2X_3)(Y_2X_3Y_3 - X_2Y_3^2) - (X_2Y_2Y_3 - Y_2^2X_3)(2Y_2X_3^2 - 2X_2X_3Y_3)},
\end{aligned}$$

where

$$\begin{aligned}
C_2 &= 4I_{12} - 3I_1 - I_2,\\
C_{12} &= I_1 + I_2 - 2I_{12},\\
C_3 &= 4I_{13} - 3I_1 - I_3,\\
C_{13} &= I_1 + I_3 - 2I_{13},\\
C_{23} &= 4I_1 - 4I_{12} - 4I_{13} + 4I_{23}.
\end{aligned}$$

The values from $T_0, \ldots, T_5$ should be substituted in equation (4) where it will be simplified to a linear function by the incremental concept. The calculation of these parameters contains 25 additions, 51 multiplications, and 5 divisions.

4. Highlight test

The method proposed in the previous section approximates the radiance by a quadratic function. If the triangles are too big and the radiance changes quickly due to a highlight, then this approximation can still be inaccurate. In order to avoid this problem, the accuracy of the approximation is estimated, and if it exceeds a certain threshold, then the triangle is adaptively subdivided into 4 triangles by halving the edges.

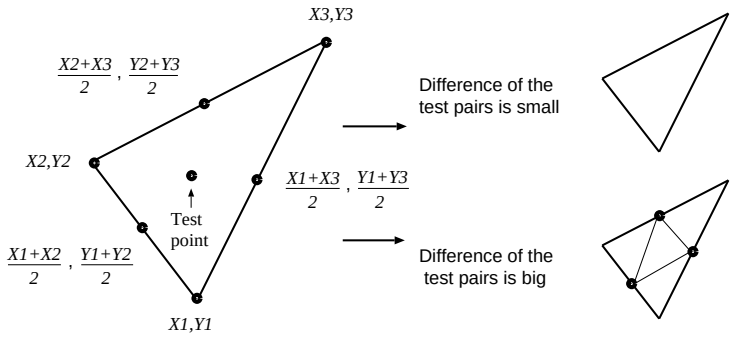

Fig. 4. Highlight test and adaptive subdivision

Recall that the knot points of the interpolation are the vertices and the middle points of the edges. Thus a reasonable point where the error can be measured is the center of the triangle. This leads to the following highlight test algorithm. Having computed the $T_0, \ldots, T_5$ parameters, the reflected radiance is estimated at the center of the triangle and the result is compared with the result of the evaluation of the shading formula. In case of big difference, adaptive subdivision takes place. Note that the overhead of one more shading evaluation is affordable.

5. Hardware implementation of the quadratic interpolation

This section reviews the implementation strategies of simple functions on scan-lines that are used to fill horizontal sided image space triangles. If the image space triangle is not formed as horizontal sided triangle, then it should be divided into two parts, a lower and an upper. In this section we will consider only the lower horizontal sided triangle. Image space triangle and horizontal sided triangle are shown in figure 5.

If we implemented equation (4) directly, the hardware should compute floating point multiplications and additions for each value, which are rather demanding. To eliminate the multiplications, we introduce the incremental concept which traces back the evaluation of the function $I(X, Y)$ to the computation of an increment from the previous

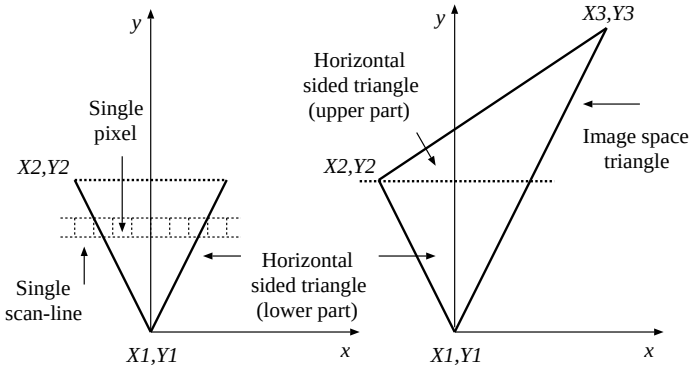

Fig. 5. Image space triangle and horizontal sided triangle

values, for instance, from $I(X-1,Y)$. The increments can then be evaluated by simple additions. The triangle filling algorithm should generate the sequence of (X,Y) integer values called pixels that are inside a horizontal sided triangle. The algorithm generates the pixels scan-line by scan-line. In a single scan-line the Y coordinate is constant. To simplify equation (4), we introduce the incremental concept for the scan-lines and for their start edges. First, the quadratic function is reduced to a linear one for the scan-lines:

$$I(X+1,Y) = I(X,Y) + \Delta I(X,Y)$$

where

$$\Delta I(X,Y) = 2T_5X + T_4Y + T_5 + T_2. \tag{5}$$

Then we apply the incremental concept once more for the linear function $\Delta I(X,Y)$ to obtain the incremental value inside the scan-line:

$$\Delta I(X+1,Y) = \Delta I(X,Y) + 2T_5.$$

When we step onto the next scan-line, Y is incremented, and the start X_{start} and the end X_{end} coordinates should be determined by the following equations:

$$\begin{aligned} X_{start}(Y) &= \frac{Y-Y_1}{Y_2-Y_1}\cdot(X_2-X_1)+X_1, \\ X_{end}(Y) &= \frac{Y-Y_1}{Y_3-Y_1}\cdot(X_3-X_1)+X_1. \end{aligned}$$

Since $X_{start}(Y)$ and $X_{end}(Y)$ are linear functions, they can be simplified by applying the incremental concept:

$$\begin{aligned} X_{start}(Y+1) &= X_{start}(Y) + A_{start}, \\ X_{end}(Y+1) &= X_{end}(Y) + A_{end}, \end{aligned}$$

where

$$A_{start} = \frac{X_2 - X_1}{Y_2 - Y_1}, \qquad A_{end} = \frac{X_3 - X_1}{Y_3 - Y_1}.$$

Now let us discuss the computation on the start edge. When the algorithm steps onto the next scan-line, both $I(X,Y)$ and $\Delta I(X,Y)$ should be recomputed with the data of the new scan-line. The incremental concept can also be used for these computations, which traces back these updates to two additions. The first application of the incremental concept reduces the computation of the quadratic function $I(X,Y)$ to a linear one:

$$I(X + A_{start}, Y + 1) = I(X,Y) + \Delta I_{start}(X,Y)$$

where

$$\Delta I_{start}(X,Y) = T_5 A_{start}^2 + (2T_5 X + T_4 Y + T_4 + T_2) A_{start} + T_4 X + 2T_3 Y + T_3 + T_1.$$

Applying the incremental concept once more for the linear function $\Delta I_{start}(X,Y)$, we obtain a constant addition:

$$\Delta I_{start}(X + A_{start}, Y + 1) = \Delta I_{start}(X,Y) + 2(T_5 A_{start}^2 + T_4 A_{start} + T_3).$$

To obtain the incremental value $\Delta I(X,Y)$ at the start edge, we should apply the incremental concept only once since it is already a linear function (equation (5)):

$$\Delta I(X + A_{start}, Y + 1) = \Delta I(X,Y) + 2T_5 A_{start} + T_4.$$

Let us group these formulae in the following algorithm:

$X_{start} = X_1$, $X_{end} = X_1$
$I_{start}(X,Y) = T_5 X_{start}^2 + T_4 X_{start} Y_{start} + T_3 Y_{start}^2 + T_2 X_{start} + T_1 Y_{start} + T_0$
$\Delta I_{start}(X,Y) = T_5 A_{start}^2 + (2T_5 X_{start} + T_4 Y_{start} + T_4 + T_2) A_{start}$
$\quad + T_4 X_{start} + 2T_3 Y_{start} + T_3 + T_1$
$\Delta I(X_{start}, Y) = 2T_5 X_{start} + T_4 Y_{start} + T_5 + T_2$
for $Y = Y_1$ **to** Y_2 **do**
  $I(X,Y) = I_{start}(X,Y)$
  $\Delta I(X,Y) = \Delta I(X_{start}, Y)$
  for $X = X_{start}$ **to** X_{end} **do**
    write(X, Y, $I(X,Y)$)
    $I(X,Y)$ += $\Delta I(X,Y)$
    $\Delta I(X,Y)$ += $2T_5$
  endfor
  $\Delta I(X_{start}, Y)$ += $2T_5 A_{start} + T_4$
  $I_{start}(X,Y)$ += $\Delta I_{start}(X,Y)$
  $\Delta I_{start}(X,Y)$ += $2(T_5 A_{start}^2 + T_4 A_{start} + T_3)$
  X_{start} += A_{start}
  X_{end} += A_{end}
endfor

Note that function I and the parameters are not integers, and if we ignored the fractional part, the incremental formula would accumulate the error to an unacceptable level. The realization of floating point arithmetic is not at all simple. Non-integers, fortunately, can also be represented in fixed point form where the low b_I bits of the code word represent the fractional part. From a different point of view, a code word having binary code C represents the real number $C \cdot 2^{-b_I}$. The number of bits in the fractional part has to be set to avoid incorrect I calculations due to the cumulative error in I. Since the maximum length of the iteration is $M = \max(Y_2 - Y_1) + \max(X_2 - X_1)$, and the maximum error introduced by a single step of the iteration is less than 2^{-b_I}, the cumulative error is maximum $M \cdot 2^{-b_I}$. Incorrect calculations of I is avoided if the cumulative error is less than 1, i.e. $b_I > \log_2 M$. Since the results are expected in integer form, they must be converted to integers at the final stage of the calculation. The round function finding the nearest integer for a real number, however, is quite difficult to implement in hardware. Fortunately, the Round function can be replaced by the Trunc function generating the integer part of a real number if 0.5 is added to the number to be converted. The implementation of the Trunc function poses no problem for fixed point representation, since just the bits corresponding to the fractional part must be neglected.

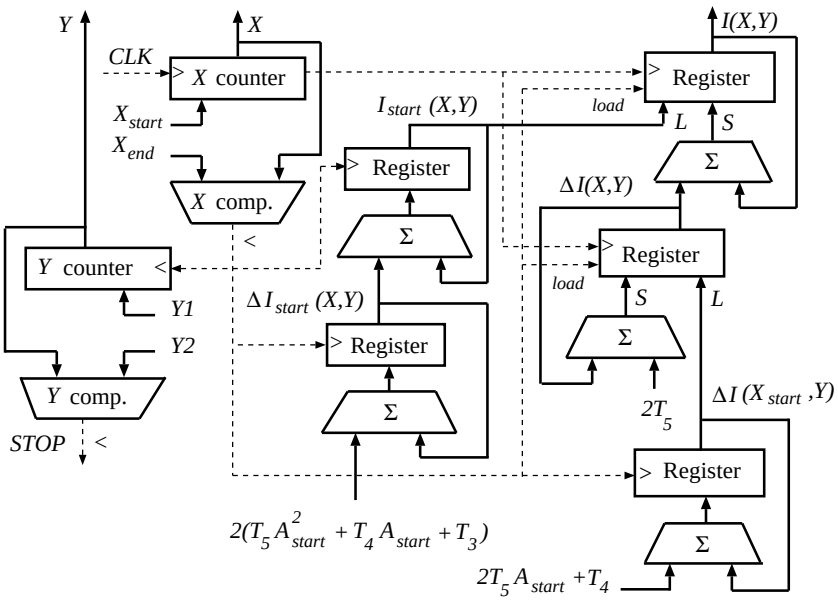

Fig. 6. Hardware implementation of two-variate quadratic functions

The hardware implementation is shown in figure 6. In figure 6 the registers usually

have two data inputs L and S. L is the input to the register when the *load* signal is active, and S is the input to the register for each clock. The clock signal of the subsystem responsible for the internal pixels of the scan-lines is the system clock. However, the clock signal controlling the elements that compute the interpolation at the start edge is the output of the comparator detecting the end of the scan-line.

6. Simulation results

The proposed algorithm has been implemented first in Microsoft Visual C++ and tested as a software. In figures 9, 10 and 11 spheres tessellated on different levels are compared. Gouraud shading evaluates the rendering equation for every vertex, quadratic shading for every vertex and edge centers and Phong shading for each pixel. The difference of the algorithms is significant when the tessellation is not very high. The measured times of drawing of the coarsely tessellated spheres are as follows: Gouraud shading 230 msec, quadratic shading 250 msec, and Phong shading 450 msec. Note that Gouraud shading performs poorly on coarsely tessellated surfaces, but the visual quality of quadratic shading and Phong shading is similar. On the other hand, concerning the speed and the suitability for hardware implementation, quadratic shading is close to Gouraud shading. Figure 8 shows a more complex scene with normal tessellation level. Looking at these images we can conclude that quadratic shading is visually superior to Gouraud shading and indistinguishable from classical Phong shading.

Having tested the software implementation, the hardware realization was specified in VHDL and simulated in ModelTech environment. The timing diagram of the algorithm is shown by figure 7. The delay times are according to XILINX XCV300-6 FPGA. In this figure we can follow the operation of the hardware. The hardware can generate one pixel per one clock cycle. The length of the clock cycle — which is also the pixel drawing time — depends on FPGA devices and on the screen memory access time. For the mentioned device it can be less than 50 nsec. While the hardware draws the actual triangle, the software can compute the initial values for the next triangle, so initialization and triangle drawing are executed parallely.

7. Conclusions

This paper proposed a new rendering strategy where the color is evaluated by the rendering equation at six representative points, three on the vertices and the other three halfway between the vertices, then it is interpolated inside the triangle according to a quadratic scheme. The algorithm has also been transformed to a hardware design that has been simulated in VHDL demonstrating that 50 nsec pixel drawing time can be obtained. Even if the screen has about 1000×1000 resolution, the complete image can be redrawn 16 times per second which provides the illusion of continuous motion.

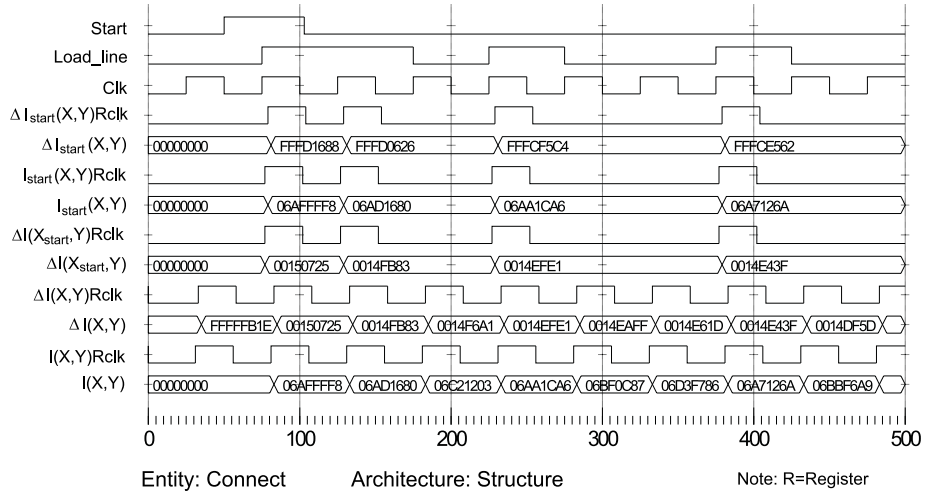

Fig. 7. Timing diagram of the hardware generated by the VHDL simulator

Fig. 8. The mesh of a chicken (left) and its image rendered by classical Phong shading (middle) and by quadratic shading (right)

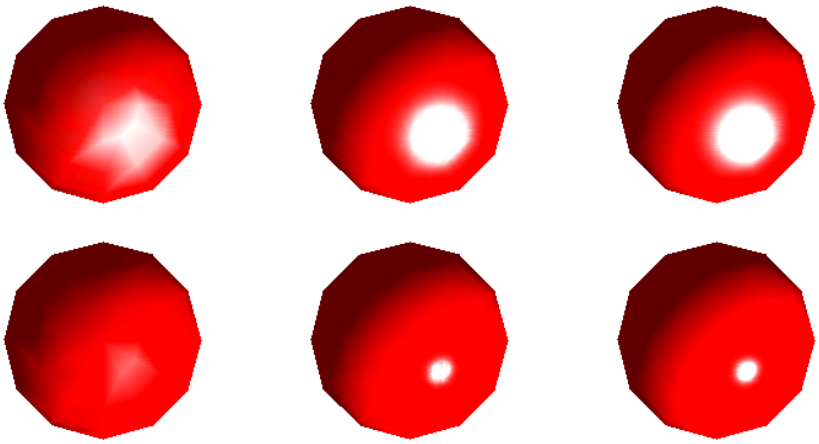

Fig. 9. Rendering coarsely tessellated spheres (168 triangles) of specular exponents $n = 5$ (top) and $n = 50$ (bottom) with Gouraud shading (left), quadratic shading (middle) and Phong shading (right)

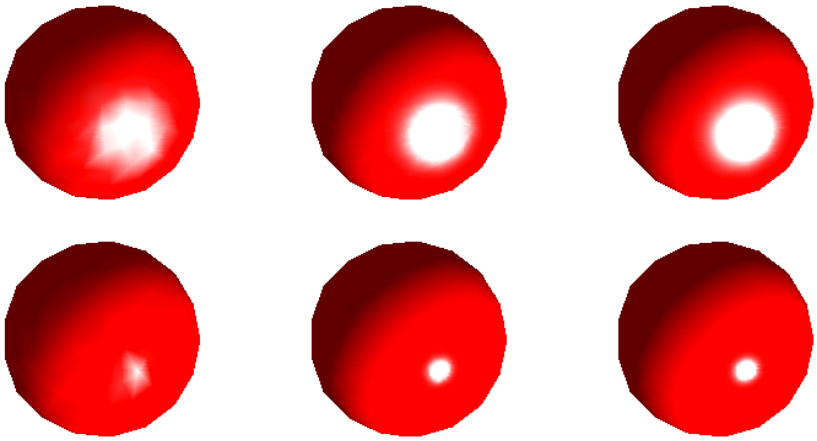

Fig. 10. Rendering moderately tessellated spheres (374 triangles) of specular exponents $n = 5$ (top) and $n = 50$ (bottom) with Gouraud shading (left), quadratic shading (middle) and Phong shading (right)

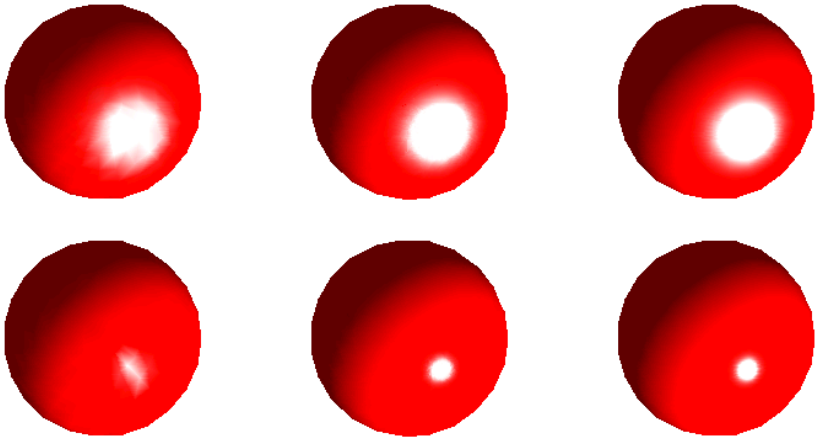

Fig. 11. Rendering highly tessellated spheres (690 triangles) of specular exponents $n = 5$ (top) and $n = 50$ (bottom) with Gouraud shading (left), quadratic shading (middle) and Phong shading (right)

References

1971
[1] H. Gouraud. Computer display of curved surfaces. *ACM Transactions on Computers*, C-20(6):623–629, 1971.

1975
[2] B. T. Phong. Illumination for computer generated images. *Communications of the ACM*, 18:311–317, 1975.

1979
[3] T. Duff. Smoothly shaded rendering of polyhedral objects on raster displays. In *Computer Graphics (SIGGRAPH '79 Proceedings)*, 1979.

1986
[4] G. Bishop and D.M. Weimar. Fast phong shading. *Computer Graphics*, 20(4):103–106, 1986.

1989
[5] A. M. Kuijk and E. H. Blake. Faster phong shading via angular interpolation. *Computer Graphics Forum*, pages 315–324, 1989.
[6] A. Watt. *Fundamentals of Three-dimensional Computer Graphics*. Addision-Wesley, 1989.

1990
[7] U. Claussen. On reducing the phong shading method. *Computer & Graphics*, pages 73–81, 1990.

1994
[8] L. Szirmay-Kalos and G. Márton. On hardware implementation of scan-conversion algorithms. In *8th Symp. on Microcomputer Appl.*, Budapest, Hungary, 1994.

1995
[9] L. Szirmay-Kalos (editor). *Theory of Three Dimensional Computer Graphics*. Akadémia Kiadó, Budapest, 1995. http://www.iit.bme.hu/~szirmay.

Ali Abbas Mohammed is a Ph.D. student at the Department of Control Engineering and Information Technology of the Budapest University of Technology and Economics. He received M.Sc. degree from the same university in 1997. His research interests include computer graphics hardware, especially the implementation of shading algorithms using high-level logic synthesis and VHDL.

László Szirmay-Kalos was graduated from the Technical University of Budapest in 1987 and received Ph.D. degree from the Hungarian Academy of Sciences in 1992. Currently he works as an associate professor at the Department of Control Engineering and Information Technology of the Budapest University of Technology and Economics where he heads the computer graphics group. His research interests include global illumination rendering, quasi-Monte Carlo techniques, visualization and object-oriented methods (homepage: http://www.iit.bme.hu/ szirmay).

Tamás Horváth is senior lecturer at the Department of Control Engineering and Information Technology of the Budapest University of Technology and Economics. He is involved in advance digital design, high-level logic synthesis and their application in computer graphics.

Tibor Fóris is a chief software engineer at the Hungarian Telematics Co. and also holds a part time position at the Budapest University of Technology and Economics. He participated in various research and development projects including software development for special purpose graphics hardware, the development of process visualisation and financial information systems.