University of West Bohemia

Faculty of Applied Sciences

Department of Computer Science and Engineering

Bachelor Thesis

Interpolation methods for Triangulation Represented Digital Image

Pilsen, 2009 Tomáš Janák

Originál zadání práce (ten s červeným kulatým razítkem) v jednom výtisku práce,
oboustranná kopie zadání (stačí černobílá) ve druhém.

I hereby declare that this bachelor thesis is completely my own work and that I used only the cited sources.

Pilsen …… ……

Tomáš Janák

Acknowledgments:

I would like to thank Ing. Josef Kohout PhD for supervising me on this project in the past two years, for his valuable insights and advices, for his quick responses to any problem which occurred and for his concerned help with their solution. Another portion of gratitude belongs to Doc. Dr. Ing. Ivana Kolingerová, for helpful discussions and advices about some of the interpolation methods, as well as for her moral support. A special thanks goes to my parents for their intensive support during my whole studies. Without them, this work surely would not be possible.

Abstract

It has been proven that triangulation can be a good tool for vector representation of image data. To visualize the image represented by triangulation, one needs to fit a continuous surface of colour intensity in the triangulation, i.e. to interpolate data stored in its vertices. The commonly used piecewise linear interpolation lacks means to adapt to behaviour of intensity in the surroundings of currently interpolated triangle. This leads to disturbance of continuity of the mentioned intensity surface. This thesis presents and describes five other interpolation methods, each representing a possibility to remove the flaws of the bilinear interpolation. Zienkiewicz's interpolation is presented as a method to use information from surrounding triangles in the form of gradient vectors; interpolations on Bézier triangle patch and on Coons patch are presented as methods of interpolation on larger surfaces; Natural Neighbour interpolation and piecewise linear interpolation on Voronoi diagram use the Voronoi diagram instead of the triangulation. Each method is described and tested and the methods are compared to each other in terms of time complexity and quality of the resulting image.

Table of Contents

1. Introduction

A digital image is nowadays usually represented by a rectangular grid of pixels, where each pixel contains a colour value or a greyscale luminance. Though this (a matrix of pixels) is the form in which we visualize the image, scaling or other transformations of such bitmap are rather problematic as they introduce various artifacts, distortions and other unwanted changes to the resulting image.

We can avoid such problems if we convert the raster into vector representation, i.e. if we represent the image as a set of geometrical simplices. The transformation is then simplified to a change of coordinates of the points, which define individual simplices (triangles in our case). We interpolate among those points to get the remaining points, which create the image. The basic strategy is to create some triangulation from the input image and, when visualizing it, reconstruct the image with a bilinear interpolation of each triangle. This simple approach obviously does not generate satisfactory results. Main problems are that large, almost mono-coloured, areas are not "smooth" enough and colour edges are not "sharp" enough. Lack of smoothness means discontinuity in colour intensity among individual triangles and lack of sharpness means that the colour edges, which make an individual object in the picture recognizable, are blurred. There are basically two directions in which this model has to be modified in order to achieve better results.

First of them is an improvement of the process of triangulation construction. The set of vertices, which forms the triangulation, resembles pixels of the original image (their x and y coordinates and colour intensity). This means that there are many different ways of how to choose which pixel should become a vertex. There are also many different types of triangulations that can be used. The construction can have various impacts on the quality of the reconstructed image, thus the choice of the type of triangulation is very important. Interpolation methods presented further in this thesis were tested on Delaunay triangulations.

The second thing to improve is the interpolation method itself, which is the topic of this thesis. Flaws of the commonly used bilinear interpolation are mainly caused by an ignorance of intensity behaviour in areas surrounding the interpolated triangle. This thesis describes some interpolation methods, which would enable us to incorporate those areas into calculations of the resulting intensity.

Chapter two briefly describes the Delaunay triangulation, the Voronoi diagram and co-triangulations, which are the geometric structures used for the representation of the image. The interpolation methods themselves are presented in chapter three. Chapter four focus on some details concerning to their implementation and singular cases. Chapter five is dedicated to image quality measurement methods, used colour systems and also the hardware and subjects configurations used for the tests are mentioned. Then the quality comparisons follow in chapter six, as well as comparisons of time consumed by individual methods. The methods are compared to each other and their overall quality is evaluated. Chapter seven summarize all the conclusions the research brought.

This main goal of this thesis is to compare interpolation approaches, which has not been examined too much so far. Though it would be great, this thesis does not present a perfect solution to the problem presented above. It is rather opening new possibilities for further development.

2. Used geometrical structures

2.1. Triangulation

Triangulations are a common tool of computational geometry not only in computer graphics, but also in various other fields, like terrain modelling, computer vision and other. We can define a common triangulation in two-dimensional space as follows [3]:

Given a point set S in two-dimensional space, the triangulation T(S) of this set of points is a set of triangles such that:

- *The point p from the space is a vertex of a triangle from T(S) if and only if p belongs to S; i.e., the vertices of the triangles are some points from the input set.*
- *The intersection of two triangles is either empty or it is a shared face or a shared vertex.*
- *The set T(S) is maximal: there is no triangle that can be added into T(S) without violating previous rules.*

However, there can be numerous different triangulations of the same point set, which match this definition. To ensure that the triangulation is unique for a given point set, additional constrains have to be set. Numerous specific triangulations, with different qualities and requirements, are known and used. For example, some research of triangulation-represented digital image using data dependent triangulations (DDT) was done by other researchers ([2], [7]). However, since this thesis is not focused on triangulations, only Delaunay triangulation (see part 2.2) will be used further, because it is suitable for our needs.

2.2. Delaunay triangulation

Delaunay triangulation (DT) was proposed by a Russian scientist Boris N. Delone. Although it can be defined for more dimensional space, only two-dimensional triangulation is considered in further text. Delaunay triangulation *DT*(*S*) of a set of points *S* in two dimensions is a triangulation such that the circum-circle of any of its triangles does not contain any other point of *S* in its interior.

Due to this additional constraint, the *DT* has the following properties [3]:

- *If no four points lie on the same circle, DT(S) is unique. E.g., four points lying in the vertices of an empty square have a common circle and two possible configurations of their triangulation.*
- *The Delaunay triangulation includes at most $O(N^{d/2})$ triangles, where N is the number of points to be triangulated.*
- *The boundary of the DT(S) is a convex hull[1] of S.*
- *DT(S) maximizes the minimal angle and, therefore, the Delaunay triangulation contains the most equiangular triangles of all triangulations (i.e., it limits the number of too narrow triangles that may cause problems in further processing).*
- *In the worst case, it can be computed in $O(N \cdot \log N)$. However, algorithms with O(N) expected time also exist.*

[1] A convex polygon made from some points from a point set S, which encapsulates all the other points in S

Because of these properties, Delaunay triangulation is used very often. Also, the Voronoi diagram (see part 2.3) is actually a dual configuration to this triangulation, which is another welcomed quality.

In our case, the Delaunay triangulation is constructed from several points, which are chosen from the original picture. Thus the resulting triangulation is actually a three-dimensional structure, where each of its vertices is defined by spatial coordinates and a colour intensity value. The points, future vertices of the triangulation, are chosen in order to preserve as much information about colour changes in the image as possible. Obviously, the more vertices the triangulation has, the more accurate the interpolation will be. More information about the construction can be found in [4].

2.3. Voronoi diagram

Voronoi diagram is a geometrical structure named after Russian mathematician Georgy Feodosevich Voronoi, who defined it. It divides the space into smaller regions, which are built around certain chosen points (vertices of the diagram). These regions, called Voronoi cells, contain such part of the space, which is closest to the respective vertex of the diagram. For two-dimensional case it can be defined as follows:

Given a point set S in two-dimensional space, the Voronoi diagram VD(S) of this set of points is a set of Voronoi cells, such that:

- *Exactly one point* p *from the point set S lies inside the cell. In further text, this point will be called the control vertex of the cell.*
- *The cell is a convex polygon.*
- *Each point* x, *for which the Euclidean distance between it and a given point* p *from the set* S *is smaller then the distance between* x *and any other point from S, i.e. control vertex of any other cell, lies inside the cell denoted by* p.
- *Each point y, for which the Euclidean distance between it and a given point* p *from the set* S, *is equal to the distance between* y *and exactly one other point* q *from the point set S, lies on an edge* e. *This edge is shared by the cells denoted by points* p *and* q. *In further text, edge* e *will be called a neighbouring edge and the two cells that share it will be called neighbouring cells.*
- *Each point* z, *for which the Euclidean distance between it and a given point* p *from the set* S, *is equal to the distance between* y *and more then one other points* R *from the point set S, is a vertex of the cell denoted by* p *and the other cells denoted by points from set* R. *In further text, all cells which share vertex* z *will be called adjacent to vertex* z *and the set of all points* z *will be called vertices of the Voronoi diagram*. *Point* z *lies on the end of neighbouring edges between neighbouring cells adjacent to* z.

Unlike the triangulation, Voronoi diagram is obviously unique for a given set of points. As mentioned in part 2.2, the Voronoi diagram is a dual configuration to the Delaunay triangulation of the same set of points. Specifically, the vertices of a given Voronoi cell with control vertex *p* are the centres of circum-circles[2] of all triangles from the Delaunay triangulation, which are adjacent to *p*. This knowledge is used to construct the diagram from the triangulation, which was the input structure used for all the implemented methods.

[2] A circle which passes thru each vertex of the triangle

2.4. Co-triangulation

Structures described in parts 2.1 – 2.3 lack the means to represent more then three values in each vertex (assuming that two of them are spatial coordinates). The main implication for purposes of digital image is that they cannot be used to represent coloured image, because commonly used colour systems, like RGB[3] for example, consist of more then one components (usually three). One possibility that allows processing of coloured images is usage of three triangulations or Voronoi diagrams, one for each colour component, interpolated separately and then merged into one coloured image. The other, more convenient way, is to use the co-triangulation.

The co-triangulation [10] is constructed from N D-dimensional data sets. These data sets are transformed into one D-dimensional Delaunay triangulation in $D + N$-dimensional space. For the purpose of coloured image representation this means that the inputs are three triangulations, one for each colour component and the output is one triangulation, which includes all the colour channels.

The spatial distribution of vertices in the input data set does not have to (and usually do not) match, i.e. for some vertex of one input set, there does not have to be a vertex at the same position in the other two sets. To be able to construct the co-triangulation, error tolerances ε_i has to be defined, one for each of the input data set. These tolerances help to direct the further construction process in order to make the resulting co-triangulation suitable for interpolation of all of its N properties.

As the process of the construction is rather complicated and actually not every interesting from the point of view of the topic of this thesis, only its main idea will be described, more details can be found in [10]. It is an iterative algorithm, starting with a simple initial triangulation and then continuing over all of the N data sets. When processing data set N_s, each vertex of the current (i.e. not final) co-triangulation is assigned some value for its s-th parameter, based on the N_s data set. The value is chosen as an average of a few points in the data set, which are nearest to the processed vertex.

After all the vertices of the current co-triangulation are processed in this way, it is interpolated and compared with the result of interpolation of the data set N_s. If the result is in bounds of the error tolerance ε_s, the data set is discarded and the algorithm continues with data set N_{s+1}. Otherwise new vertices are added to the co-triangulation, in order to satisfy the error tolerance. The new vertices are chosen directly from the data set N_s, i.e. the new vertex has the same spatial coordinate and the same value in the s-th parameter. For $s > 1$, there are already some other processed parameters which has to be assigned to the new vertex. In this case, these parameters are gained by bilinear interpolation of the triangle from the former co-triangulation (i.e. the one before the insertion of the new point), in which the inserted point lies.

For more insight and details about the algorithm and the co-triangulation itself, please refer to [10]. The important conclusion for the case of this thesis is that the co-triangulation is a structure, which allows processing of coloured digital image in exactly the same way as the usual triangulations – the resulting structure is also a Delaunay triangulation, only with five (e.g. x, y, R, G, B) instead of three (x, y, intensity) parameters assigned to each vertex.

[3] Each colour is described as a combination of Red, Green and Blue component

3. Interpolation methods

3.1. Bilinear interpolation

Bilinear interpolation is the most commonly used method for interpolating a triangle. This popularity is based mainly on its simplicity, both in terms of computational time and its implementation. Its idea is very simple: first, linearly interpolate pixels on the edges of the triangle and then use these values to linearly interpolate each horizontal line of the triangle. To simplify this algorithm, we can use barycentric coordinates to achieve the same results without having to distinguish between pixels on the edge and inside the triangle.

Barycentric coordinates of a triangle are three numbers respective to individual vertices of the triangle. They describe every point inside the triangle as an interpolation of the vertices. Equation 3.1 shows how they are computed.

$$a = \frac{Area(BCP)}{Area(ABC)}, \quad b = \frac{Area(ACP)}{Area(ABC)}, \quad c = \frac{Area(ABP)}{Area(ABC)}$$

Equation 3.1: Barycentric coordinates a, b, c of point P in triangle ABC. They are gained as a portion of triangle created by point P and the two vertices, which are opposite to the vertex for which the coordinate is computed. Therefore coordinates a, b, c belongs to vertices A, B, C respectively.

A simple linear combination of the barycentric coordinates and their respective vertices is then the Bilinear interpolation. This allows us to interpolate the colour intensity of all pixels using the following algorithm:

```
For each triangle ABC of the triangulation do
     Area = compute_area(ABC);
     For each pixel P of the triangle do
          a = compute_area(BCP) / Area;
          b = compute_area(ACP) / Area;
          c = compute_area(ABP) / Area;
          P.Intensity = a * A.Intensity + b * B.Intensity + c
     * C.Intensity;
```

3.2. Zienkiewicz's interpolation

This method was used in [1] for interpolation of geographical data represented by triangulation. Because of its plausible results, it was worth trying to exploit it for our purpose. It uses gradient vectors to describe the behaviour of the intensity in the area that surrounds the interpolated triangle.

Gradient in a vertex can be estimated as an average of normalized surface normal vectors (which are gained as a cross product of any two edge vectors of a triangle) of each of the triangle adjacent to the vertex, weighted by their areas, as expressed by Equation 3.2.

This estimation is then inserted into formula of Zienkiewicz's interpolation, which cubically interpolates intensity across the triangle. The formula is described by Equation 3.3.

$$normal = \sum_k \frac{A_k \cdot n_k}{A} \qquad gradient = \left(\frac{-normal_X}{normal_I}, \frac{-normal_Y}{normal_I} \right)$$

Equation 3.2: Estimation of a gradient. n_k *is the normalized surface vector of the* k*-th triangle,* A_k *is its area and* A *is the sum of all the areas* A_k*. With the weighted normal, gradient along x and y axis is estimated as the* x *and* y *component of the normal divided by its intensity component* $normal_I$.

$$u_1 = A_i \qquad u_4 = B_i \qquad u_7 = C_i \qquad k_1 = 2 \cdot a \cdot b \cdot c$$
$$u_2 = AB \cdot g_a \qquad u_5 = BC \cdot g_b \qquad u_8 = \overline{CA} \cdot g_c \qquad k_2 = \frac{a \cdot b \cdot c}{2}$$
$$u_3 = \overline{AB} \cdot g_b \qquad u_6 = BC \cdot g_c \qquad u_9 = CA \cdot g_a$$

$$\begin{aligned} P(a,b,c) = {} & u_1\left[a^2 \cdot (3-2a) + k_1\right] + u_2 \cdot (a^2 \cdot b + k_2) - u_3 \cdot (a \cdot b^2 + k_2) + \\ & + u_4\left[b^2 \cdot (3-2b) + k_1\right] + u_5 \cdot (b^2 \cdot c + k_2) - u_6 \cdot (b \cdot c^2 + k_2) + \\ & + u_7\left[c^2 \cdot (3-2c) + k_1\right] + u_8 \cdot (c^2 \cdot a + k_2) - u_9 \cdot (c \cdot a^2 + k_2) \end{aligned}$$

Equation 3.3: Zienkiewicz's interpolation. For better clarity, some terms of the actual formula were shortened into terms $u_1 - u_9$ *and* k_1 *and* k_2*.* AB, BC *and* CA *are the edge vectors of the triangle ABC (edges* B - A, C - B *and* A - C*, respectively),* g_a, g_b *and* g_c *are gradients in vertices* A, B, C*;* a, b, c *are the barycentric coordinates and* A_i, B_i, C_i *are colour intensities in the corresponding vertices. The formula itself is a function of the barycentric coordinates of point* P.

Result of the interpolation formula described in equation 3.3 is the intensity value in the interpolated pixel. Therefore the algorithm for interpolation of the whole triangulation is similar to that of a bilinear interpolation; we only need to compute the gradient vectors beforehand:

```
For each triangle ABC of the triangulation do
     ComputeGradient(ABC);
For each triangle ABC of the triangulation do
     u1 = A.Intensity;
     u2 = (B.x - A.x) * A.grad.x + (B.y - A.y) * A.grad.y;
          ...
     u9 = (A.x - C.x) * A.grad.x + (A.y - C.y) * B.grad.y;
     For each pixel P of the triangle do
          GetBarycentricCoo(out a, out b, out c, ABC);
          k1 = 2 * a * b * c, k2 = k1 / 4;

          P.Intensity = u1 * (a^2 * (3 - 2 * a) + k1) + u2 *
     (a^2 * b + k2) - u3 * (a * b^2 + k2) + ... - u9 * (c * a^2 +
     k2);
```

3.3. Interpolation on the Bézier triangle patch

Bézier triangle patch is a triangular surface, which can be made by joining three Bézier curves[4]. Its degree is denoted by the degree of these curves, i.e. by the number of control points the curves have. For our purpose we consider patches of the second degree. Therefore, each boundary curve has three control points, which makes six control points to define the surface. A possible configuration of such a patch can be seen in Figure 3.1.

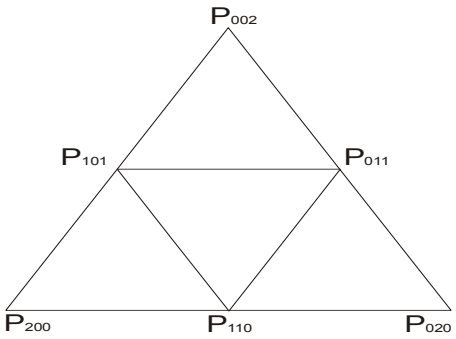

Figure 3.1: Bézier patch of the second degree with its control points marked $P_{200} - P_{002}$.

Upon looking on Figure 3.2, another way of interpretation of such a patch might occur. It can be viewed it as a patch created by some triangle and its neighbours, i.e., the triangles, which share an edge with the one in centre. Therefore, the patch can be easily constructed for each triangle in our triangulation, by appending the neighbours of the triangle to it.

To describe points inside the patch, barycentric coordinates of vertices P_{200}, P_{020} and P_{002} can be used. They will be referred to as *a, b* and *c,* respectively. The conditions ensuring that the point *P(a, b, c)* lies inside triangle P_{200}, P_{020}, P_{002} are mentioned in Equation 3.4.

$$a + b + c = 1 \quad \wedge \quad a \geq 0; \;\; b \geq 0; \;\; c \geq 0$$

Equation 3.4: Constrains for barycentric coordinates to describe points inside a triangle.

Point *P* lying in the patch can than be described in the form of Bernstein polynomials with the barycentric coordinates as parameters, as Equation 3.5 shows:

$$P(a,b,c) = \sum_{i+j+k=n} B_{ijk}^{n}(a,b,c) \cdot P_{ijk}$$

$$\text{where} \quad B_{ijk}^{n}(a,b,c) = \frac{n!}{i! \cdot j! \cdot k!} a^i b^j c^k$$

Equation 3.5: Bernstein polynomials describing point P *as a function of its barycentric coordinates* a, b, c. n *is the degree of the patch.*

After inserting our values into Equation 3.5, i.e. the degree $n = 2$ and expanding it, we get Equation 3.6.

[4] Curves defined as a set of points, where each point is assigned several parameters to control the derivatives of the curve. They became very popular for purposes of geometrical modelling (originally for car industry, by Pierre Bézier, 1962).

$$P(a,b,c) = a^2 \cdot P_{200} + b^2 \cdot P_{020} + c^2 \cdot P_{002} + 2ab \cdot P_{110} + 2bc \cdot P_{011} + 2ac \cdot P_{101}$$

Equation 3.6: Formula of Bézier triangle patch of second degree. a, b, c *are barycentric coordinates of vertices* P_{200}, P_{020} *and* P_{002}.

With the resulting formula presented in Equation 3.6, we are able to describe the whole surface defined by our patch. Coordinates (x, y and intensity) of each point P_{ijk} of this surface are obtained from Equation 3.6 by a successive insertion of x, y and intensity coordinates of respective points P_{ijk} into the formula.

All points defined by coordinates *a, b, c*, which satisfy conditions in Equation 3.4, then form a surface with the following qualities:

- It lies in the convex hull of all control points of the patch.
- It passes through terminal control points P_{200}, P_{020}, P_{002}.
- Its boundaries, thanks to the quadratic interpolation function, are parabolic splines corresponding to the control points of individual boundary curves.

Though these properties are welcome, they also cause problems with determining which pixels actually belong to the patch. The points of the patch do not have to project into each pixel of the triangles, which create the patch. On the contrary, some of those points can project outside of those triangles. Therefore it is unknown how many pixels, and which of them, will be in the patch, before the computation is made. The algorithm for the interpolation will also be different, because there cannot be any "for each pixel" loop.

A proposed solution to this problem is to compute intensity values for all possible triplets *a, b, c* that satisfy Equation 3.4. Of course, it is not possible to count with every number between zero and one. But since the number of the pixels to be rendered is finite and relatively small, a sufficiently large set of coordinates can be computed. This can be done if any two of the coordinates are subsequently incremented from zero to one by some "step", a small (much smaller than one) real number. The third coordinate is easily calculated, because, as Equation 3.4 shows, their total sum must be one. This way the desired surface can finally be rendered.

However, the choice of this "step" brings some difficulties. Real values (*a, b, c*) are projected on integer values (pixel coordinates) and therefore some pixel could be missed entirely because of rounding mistakes. Therefore, even if an approximation of the number of pixels in the patch could be calculated, there is no insurance that it will suffice to compute that many intensity values. Much more values than there are pixels to render have to be computed, which result in lengthier time consumed by the computation. Though some optimization can be made (e.g. the mentioned estimation of pixel count), the quadratic time complexity of the algorithm makes it unable to compete with interpolation methods that interpolate on a single triangle, like bilinear or Zienkiewicz's method, mentioned in parts 3.1 and 3.2.

On the other hand, each of the redundant value always projects onto some pixel. Because of that, more and not necessarily equal intensity values for each pixel are obtained. Note that this is not the only reason why there are more values for each pixel – the patches are constructed for each triangle in the triangulation. And because each patch is made of four

triangles, each triangle is involved in four patches. This implies even more intensity values for pixels in the intersection of the neighbouring patches. It must be ensured that the finally displayed value is correct (that it did not end in the pixel only due to a rounding mistake). Theoretically, the most often value should be the most suitable, but experiments showed that results are almost the same as with the arithmetical average of all values projected into the pixels. Because computing arithmetical average has much lower memory demands, it is the suggested option.

Finally, the algorithm for interpolation using the Bézier triangle patch can be stated:

```
Bitmap = array with length equal to the number of pixels,
     filled with 0 /* for a coloured image, there would be
     three such arrays, one for each colour component */
Counter = array of the same length as Bitmap, filled with 0
For each triangle ABC of the triangulation do
     a = 0, b = 0;
     P200 = Vertex of the first neighbour, which is not also
a vertex of ABC
     P020 = Vertex of the second neighbour, which is not also
a vertex of ABC
     P002 = Vertex of the third neighbour, which is not also
a vertex of ABC
     P110 = A, P011 = B, P101 = C;
     step = 0.005; /* the step can be various */
     While a <= 1 do
          a = a + step;
          While b <= 1 - a do
               b = a + step;
               c = 1 – a – b;

               X = a^2 * P200.X + b^2 * P020.X + c^2 * P002.X + 2
          * a * b * P110.X + 2 * b * c * P011.X + 2 * a * c *
          P101.X;
               Y = a^2 * P200.Y + b^2 * P020.Y + c^2 * P002.Y + 2
          * a * b * P110.Y + 2 * b * c * P011.Y + 2 * a * c *
          P101.Y;
               I = a^2 * P200.Intensity + b^2 * P020.Intensity
          + c^2 * P002.Intensity + 2 * a * b * P110.Intentsity
          + 2 * b * c * P011.Intensity + 2 * a * c *
          P101.Intensity;

               Bitmap[Y * Width + X] += I;
               Counter[Y * Width + X]++;

i = 0;
While i < length of Bitmap do
     Bitmap[i] = Bitmap[i] / Counter[i]
     i++;
```

3.4. Interpolation on the Coons patch

Coons patch[5] is a surface defined by four curves as boundaries of the patch. As in the case of the Bézier triangle patch (part 3.3), this method suggests creating the Coons patch from quartets of triangles (one central triangle and his neighbours). The boundary curves are then defined by vertices on edges of such a configuration, i.e. P_{200}-P_{110}-P_{020}, P_{020}-P_{011}-P_{002} and P_{200}-P_{101}-P_{002} (using notation as presented in Figure 3.1). In order to get four curves, subdivision of the longest of those three is suggested. But before that, decision about what kind of curves will actually lead through those vertices must be made.

There are three control points per curve. Therefore it is reasonable to define the boundary curves for the patch as parabolic, i.e. defined by a quadratic polynomial. Equation 3.7 shows a parametrical formula for such a curve (with the parameter *t*).

$$curve(t) = a \cdot t^2 + b \cdot t + c$$

Equation 3.7: Parametrical formula of a parabolic curve. t *is the parameter, coefficient* a, b, c *define behaviour of the curve.*

In order to set the curve uniquely, coefficients *a, b* and *c* are found in order to fulfil this requirement: the curve must pass through the three vertices on each border of the triangle configuration. Simply said, the terminal vertices *A* and *B* (which are equal to either P_{200} and P_{020}, P_{020} and P_{002} or P_{200} and P_{002}) will be the "start" and "end" of the curve and the middle vertex *M* (P_{110}, P_{011} or P_{101}) will be the peak (or at least close to the peak) of the created parabola. This requirement can be written as a system of equations, presented as Equation 3.8, for parameter *t* varying from zero to one. Its solution is the three coefficients *a, b, c* to be inserted into Equation 3.7.

$$\begin{aligned} A &= curve(0) = a \cdot 0 + b \cdot 0 + c \\ M &= curve(0.5) = a \cdot 0.25 + b \cdot 0.5 + c \\ B &= curve(1) = a \cdot 1 + b \cdot 1 + c \\ &\text{----------} \\ a &= 2 \cdot A + 2 \cdot B - 4 \cdot M \\ b &= 4 \cdot M - 3 \cdot A - B \\ c &= A \end{aligned}$$

Equation 3.8: Solution of Equation 3.7 for parabolic curve. A *and* B *are its terminal vertices,* M *is the peak.*

Note that *A, B* and *M* represent either x, y or the intensity value of the corresponding vertex, therefore, each curve is actually defined by nine coefficients – a_x, b_x, c_x; a_y, b_y, c_y; a_I, b_I, c_I.

To select which curve to subdivide, distances from *A* to *M* and *M* to *B* are computed for all three curves and the longest curve is then subdivided into two as follows: x, y and intensity values of the longest curve for $t = 0.25$ and $t = 0.75$ are computed using formula Equation 3.7 (with *a,b,c* coefficients already known). The results are used to construct vertices $T_{0.25}$ (for $t = 0.25$) and $T_{0.75}$ (for $t = 0.75$). Vertices *A,* $T_{0.75}$, *M* then define one of the new curves and *M,* $T_{0.25}$, *B* the other.

[5] Defined by Steven Anson Coons in 1964

In this way we get two pairs of opposite curves. We can now use them to describe the surface delimited by them. Intensity value in each point of the surface is gained as a superposition of splines corresponding to the boundary curves in that point characterized by some parameter.

The routine then looks subsequently: let a_1, a_2 and b_1, b_2 be the pairs of the opposite boundary curves. Values in the parameter u for the curves b_1, b_2 gives us terminal points of the spline b_u. In Figure 3.2, the terminal points are marked as P_0 and P_{10}. We interlace this spline with each possible spline a_v characterized by the points of pair a_1, a_2, i.e. by each parameter v from zero to one. Parameter u is then incremented (a new spline b_u is chosen) and the inner cycle repeats. Routine ends when u reaches one (every possible spline b_u has been used).

In Figure 3.2, two iterations of this routine with step 0.1 are depicted. During each iteration, you can see the spline b_u (marked by a solid line) being interlaced with all eleven splines a_v with v varying from zero to one (marked by a dashed line). Each intersection of those splines is the interpolated point $P(u, v)$.

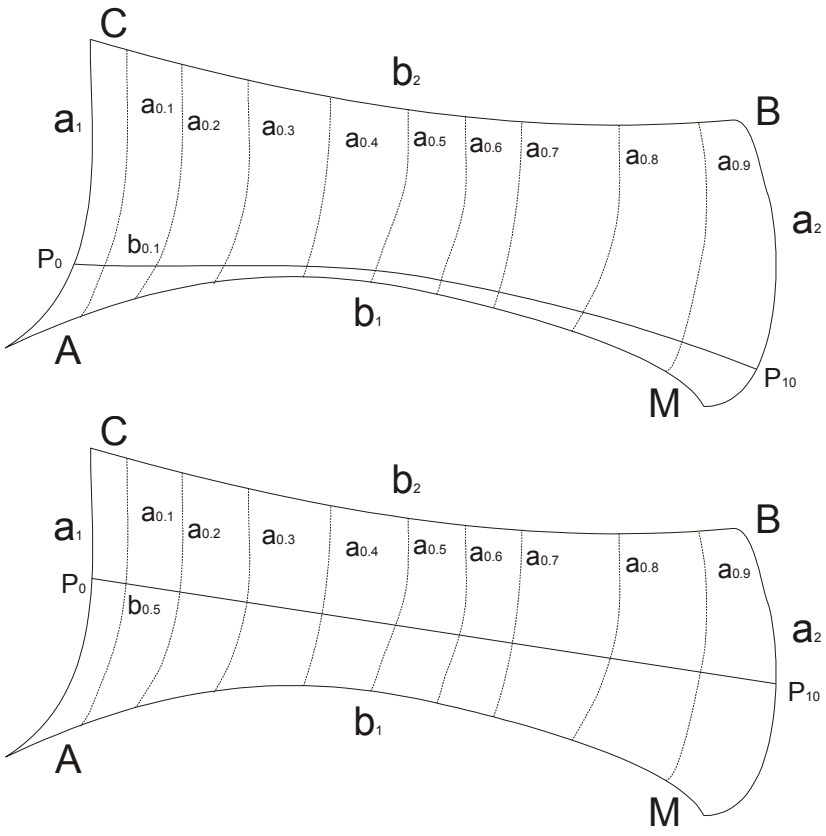

Figure 3.2: Visualization of two iterations of the spline superpositioning. A, B, C *and* M *are the terminal vertices of the boundary curves* a_1, a_2, b_1 *and* b_2.

As in the Bézier patch interpolation, it is unknown which pixel lies on the surface. Therefore the "step" we add to the parameters during each iteration of the cycle should be as small as possible to ensure that all pixels are covered. Computation of a point of the surface (the superposition of splines in a certain point) can be expressed by Equation 3.9.

$$\begin{bmatrix} 1-u & -1 & u \end{bmatrix} \cdot \begin{bmatrix} A & a_1(v) & M \\ b_1(u) & P(u,v) & b_2(u) \\ C & a_2(v) & B \end{bmatrix} \cdot \begin{bmatrix} 1-v \\ -1 \\ v \end{bmatrix} = 0$$

Equation 3.9: Formula expressing the spline superposition depicted on Figure 3.3. A, M, B and C *are spatial or colour intensity values in the respective vertices,* $a_1(v)$, $a_2(v)$, $b_1(u)$, $b_2(u)$ *are spatial or colour intensity values of points in respective boundary curves, denoted by parameters* u *and* v. *The curves and vertices used here are described in Equation 3.11.*

After expansion of Equation 3.9, shown in Equation 3.10, *x* and *y* coordinates as well as the colour intensity of point *P* dependently on parameters *u* and *v* can be obtained.

$$P(u,v) = (1-u)\cdot a_1(v) + u\cdot a_2(v) + (1-v)\cdot b_1(u) + v\cdot b_2(u) +$$

$$-(1-u)\cdot(1-v)\cdot A - u\cdot(1-v)\cdot C - (1-u)\cdot v\cdot M - u\cdot v\cdot B$$

Equation 3.10: The interpolated point P *as a function of parameters* u, v *of the boundary curves. This formula is gained by extraction of the term* P(u,v) *from Equation 3.9.*

Control points of the curves a_1, a_2, b_1, b_2 are described in Equation 3.11. The considered longest initial curve is the curve denoted by the vertices *A, M, B*.

$$a_1 = (A, M_{AC}, C),\ a_2 = (M, T_{0.25}, B)$$

$$b_1 = (A, T_{0.75}, M),\ b_2 = (C, M_{BC}, B)$$

Equation 3.11: Boundary curves a_1, a_2, b_1, b_2, *described by their terminal and middle vertices.* M_{AC} *is the vertex between* A *and* C, M_{BC} *is the vertex between* B *and* C, $T_{0.25}$ *and* $T_{0.75}$ *are vertices created during subdivision of curve* A, M, B.

The visualized value in each pixel is also gained as an arithmetic average of all values which were projected into that pixel (see part 3.3 for details behind this decision).

Unfortunately, among other similarities, this method also shares a long computing time with the interpolation on the Bézier triangle patch. The reason is the same algorithm that is used for intensity distribution among pixels in the patch. The full algorithm of the interpolation is similar to that of Bézier triangle patch interpolation, only with different equations for the interpolation. The only significant change is the computation of the boundary curves, which is described by Equations 3.7 and 3.8 and is neither interesting nor complex in terms of the algorithm. Therefore the description of the interpolation algorithm in pseudo code will be omitted in this part.

3.5. Piecewise linear Interpolation on the Voronoi diagram

This method is loosely based on usual bilinear interpolation, which uses barycentric coordinates of the pixel in a triangle as weights for the interpolation. The same principle is applied in this method, only this time it uses Voronoi cell instead of a simple triangle. The cell is a polygon with generally more then three edges, which implies the main problem

associated with this method – finding barycentric coordinates of a general convex polygon is much more complicated then finding them for a triangle.

However, this problem has already been solved and an easy way of computing barycentric coordinates for a convex polygon was described in [Mark Meyer, Haeyoung Lee]. Equation 3.12 describes the proposed formula for a barycentric coordinate in a given vertex of the polygon. Such coordinates retain all the useful properties as common triangular barycentric coordinates. Namely, as in Equation 3.4, their sum is one and they are positive for points inside the polygon.

$$\omega_j = \frac{\cot(\gamma_j) + \cot(\delta_j)}{\| P - Q_j \|^2}; \quad B_j = \frac{\omega_j}{totalSum}; \quad totalSum = \sum_{j=0}^{N} \omega_j$$

Equation 3.12: barycentric coordinate of point P respective to vertex Q_j in a convex polygon. γ and δ are angles between lines P-Q_j, Q_j-Q_{j-1} and P-Q_j, Q_j-Q_{j+1}. B_j is the resulting coordinate.

Complications occur when the interpolated point is too close to edges of the cell. Such cases are checked and handled specially – linear interpolation of the vertices of the edge where the interpolated pixel lies is used instead. Simple test of parallelism (cross product) between vectors of the currently involved edges and the vector of the interpolated point and the point Q_j can reveal these cases.

```
For each cell C of the Voronoi diagram do
     For each pixel P of the cell C do
          SumOfCoordinates = 0;
          Coo = array of barycentric coordinates;
          For j = 0, while j < number of vertices of C do
               A = vector P - C[j];
               PREV = vector C[j-1] - C[j];
               NEXT = vector C[j+1] - C[j];
               TEST = cross product of A and PREV;
               IF TEST == 0
                    P.Intensity = Linear interpolation
               between C[j-1] and C[j];
                    Continue with next pixel;
               TEST = cross product of A and NEXT;
               IF TEST == 0
                    P.Intensity = Linear interpolation
               between C[j+1] and C[j];
                    Continue with next pixel;
               C_PREV = cotangent between A and PREV;
               C_NEXT = cotangent between A and NEXT;
               Coo[j] = (C_PREV + C_NEXT) / (A.X2 + A.Y2);
               SumOfCoordinates = SumOfCoordinates + Coo[j];
          Divide each coordinate by SumOfCoordinates;
          P.Intensity = 0;
          For j = 0, while j < length of Coo do
          P.Intensity += Coo[j] * C[j].Intensity;
```

3.6. Natural neighbour interpolation

This interpolation method was taken from [5] and only transformed for the needs of a digital image. It uses two Voronoi diagrams – the one on the input (i.e. created from the triangulation that is on the input) and other, which contains the interpolated point as a vertex of the diagram. The new vertex locally modifies the initial diagram – some cells next to the point will become smaller, because the new cell, created around the new vertex, "steals" some area of the adjacent cells. The main idea of the interpolation is to compute this stolen area and use it to assign weights to the vertices, from which it was stolen. Those vertices then participate in the interpolation.

The main problems are to construct the second diagram and to find the participating cells. Luckily, this is actually one and the same if we use the incremental insertion algorithm for creation of the new diagram. First, an arbitrary participating cell has to be found. The cell, in which the point that is going to be inserted lies, definitely fulfils this demand, so it can be used as an initial cell of the process. The insertion algorithm in one cell can be described in the following steps:

> *1) Construct a line between the inserted point and the control point of the participating cell.*
> *2) Find the centre of this line and construct a parallel line, passing thru the centre.*
> *3) Find the intersections A and B of the parallel line and edges of the cell. Line AB is an edge of the new cell.*

The edges on which are the points A and B are shared with other two cells. And as points A and B are part of the new cell, these another cells must also be the participating cells. Therefore the same algorithm can be used on one of these cells and another edge of the new cell will be gained. One end of this edge will be the old intersection B (or A), and second end will be another point, which will also reveal another participating cell. The algorithm continues this way until it reaches the initial cell from another end, i.e. the newly found intersection will be the second intersection from the initial cell. This way the new cell is obtained as well as all the participating cells.

The weights assigned to each of the participating cells are gained as a fraction of the stolen area and the whole area of the new cell. Therefore, the bigger is the area stolen from a cell, the bigger is the impact this cell has on the interpolation. The resulting colour is gained as a linear combination of the control points of the cells and the weights assigned to them.

There are singular cases of the insertion (the parallel line from step 2 passes directly thru a vertex, or is identical to an edge), which needs to be taken care of in the implementation. However, they do not need any special treatment, like for example parallel vectors in the method described in part 3.5 and therefore they are mentioned only in part 4.4 (as well as other implementation problems of this method).

```
For each cell C of the Voronoi diagram do
     For each pixel P of the cell C do
          TotalArea = 0;
          Areas = list of the areas stolen from cells;
          Cells = list of participating cells;
          Used = list of already used participating cells;
          Add C to Cells;
```

```
While Cells is not empty do

     StolenCell = polygon stolen by P from cell at
index 0 in Cells;
     Add the area of StolenCell to Areas;
     TotalArea = TotalArea + the area of
StolenCell;
     Add neighbouring cells of Cells[0] which share
the dividing edge of StolenCell to Cells unless
they are in Used;
     Add Cells[0] to Used;
     Remove cell at index 0 from Cells;

Divide each area by TotalArea;
P.Intensity = 0;
For j = 0, while j < length of Coo do
     P.Intensity += Areas[j] * Used[j].Intensity;
```

4. Implementation difficulties

4.1. Bilinear and Zienkiewicz's interpolation

The Bilinear interpolation, as described in part 3.1, and the Zienkiewicz's interpolation (part 3.2) are both very easy to implement and also very similar in terms of the used algorithm. There is only one problem related to both of these interpolation methods.

When interpolating using these methods, a pixel is the basic unit – for each pixel of a triangle, the colour intensity is set. However, the edges of the triangle have no width – they are not "pixelated". As such, they can divide the pixels they intersect into two pieces, each belonging to one of the neighbouring triangles, which share the intersecting edge. The choice of which of the neighbouring triangles should be the one to use for computing the intensity of the pixel actually should not be important, because the interpolation should be smooth over triangle edges anyway. The barycentric coordinates are the real problem. Their computation, as presented in part 3.1, assumes that the interpolated point is inside the triangle, or at least on its edges. But the coordinate that is passed to the formula is the coordinate of a pixel, an integer.

Figure 4.1 shows what this integer means in the space of real numbers. It is not the whole pixel anymore; it is the upper left corner of the pixel. The point 1;2 on the figure clearly does not lie inside the marked triangle. But the pixel 1;2 does, though only partly, and therefore the interpolation algorithm will use it. If the coordinates 1;2 are inserted into the formula for computing the barycentric coordinates, the results will not meet the constraints for points inside triangle (Eq. 3.4), i.e. some of the resulting barycentric coordinates might get bigger then one, or smaller then zero.

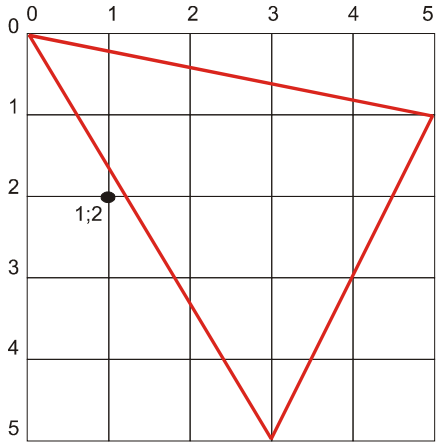

Figure 4.1: a triangle on a 5x5 grid of pixels. In the space of real numbers, pixel coordinates are actually the coordinates of their upper left corners. Therefore, points denoted by pixel coordinate, e.g. 1;2, may not actually be inside the triangle, though the pixel 1;2 is.

Using these coordinates for intensity computation may result in colour intensity being out of the range of the colour system (i.e. over 255, when using standard 8-bit per colour component format), or negative. Either way, it won't be possible to display such colour. A simple solution is to skip interpolating the pixel when the barycentric coordinates does not meet the constraints in Eq. 3.4. This way it is ensured, that no fault colour intensities will be set to the pixels. On the other hand, because the triangulation covers the whole image, it

is ensured that all the pixels will eventually be set – for each pixel there is a triangle which contains its upper left corner. At the same time, this solution does not require much more of additional code or calculations.

4.2. Patch-based interpolations

Parts 3.3 and 3.4 described two interpolation methods, which interpolate on a patch constructed from a triangle and its three neighbours. But on the edges of the triangulation, which is usually shaped as a rectangle, there are triangles which obviously have only two neighbours, because their third edge lies on the edge of the triangulation. Moreover, in the corners of the triangulation there might be triangles with only one neighbour. Therefore, the methods cannot be used in the same way as described in the third chapter.

The only way to solve this problem is to use a different interpolation method, which interpolate on a single triangle, for these triangles. For example, the solution used for experiments presented in chapter six interpolates the bitmap using bilinear interpolation first. Therefore, the array "Bitmap", as used in the algorithm description in the end of part 3.3, will be filled with valid colour intensity values for each pixel. The interpolation then continues as usual, adding the computed intensity values for each pixel together and counting them. In the end, when the average of these values is made, it is assumed, that the number of values in each pixel is one more then in the "Counter" array; that is the one value from the pre-processing.

This way, pixels on the borders of the image will have their intensity assigned as if they were interpolated by bilinear interpolation, as there will be zero values computed by the patch-based method and one value computed by bilinear interpolation. The rest of the image will be slightly modified by the result of the linear interpolation, but if the stepping of the patch-based method (either Coons of Bézier) is dense enough, it will not be significant, as there will be approximately other twenty values in each pixel, computed by the patch-based method.

4.3. Interpolation on Voronoi diagram

The first difficulty connected with interpolation on Voronoi diagram is the construction of the diagram itself. Part 2.3 mentions that the diagram can be created from the Delaunay triangulation: each vertex of the triangulation is a control vertex of a Voronoi cell and each centre of a circum-circle of each triangle is a vertex of the Voronoi diagram. Although various ways to compute locations of the centres of circum-circles are known and easy to implement, the colours in the vertices has to be interpolated somehow.

The implementations used for experiments presented in chapter six uses Zienkiewicz's interpolation to interpolate the pixels corresponding to the location of Voronoi diagram's vertices and sets their colour value this way. Nevertheless, it is important to realize that this way, an error in the colour intensity of the image is introduced prior to the interpolation itself (which brings another error to it). Better results might be obtained if the colour is set according to the original image. However, this image is not always available (after all, the main role of the triangulation is to represent that image) and it is over the topic of this thesis to exploit this possibility.

Another problem is also bound to the centres of circum-circles. For the triangles on the edges of the image, there is nothing that would assure that their centres of circum-circles

will be exactly on the edge, or over it. Therefore the Voronoi diagram does not have to cover the whole image, which would lead to leaving parts of the image blank after its interpolation. This is certainly not tolerable. It is necessary to enclose the diagram in order to cover the whole image.

To do that, three artificial vertices are added to the triangulation. These vertices form a huge triangle, which encapsulate the whole triangulation by connecting the vertices on the edges and the artificial vertices – several artificial triangles are created. For example, for 512x512 points large triangulation, vertices with coordinates (256, -10000), (-10000, 10000) and (10000, 10000) are added.

When the Voronoi diagram is created from this new triangulation, it is assured that whole image is covered, because the new cells on the border of the image span far over it. The remaining action is to clip the diagram by the edges of the image and this way, a Voronoi diagram covering whole image is gained.

Unfortunately, another problem occurs here. The vertices on the edges, created during clipping, do not have any colour intensity value associated to them. In the presented implementation, the value assigned to them is the value in the nearest vertex of the cell – that is the second vertex on the same edge as the clipped one. Although in some cases this is quite precise approximation, in some cases it is not. Therefore the results on the edges of the image tend to be extremely poor for the piecewise linear interpolation on the Voronoi diagram.

The natural neighbour interpolation does not use the colour intensity values in the cell's vertices to get the interpolated value; it uses the value in the control vertices of the cells. These are actually vertices of the triangulation, so their intensity value is correct. However, the cells created by clipping do not have to meet the definition of a Voronoi cell, i.e. some of the points in one of such cell might actually be closer to the control vertex of some other cell. Therefore the areas of these cells, which are used as weights for the interpolation, are degraded to a set of random numbers and quality of the resulting interpolation obviously decreases.

In the presented implementation, the Cohen-Sutherland clipping algorithm [11] was used for clipping. It was slightly modified, as this algorithm was originally designed for line clipping. However, clipping of cells covering the corners is practically impossible without using more complicated algorithm. As was said in the previous paragraphs, the vertices created by clipping introduce some error to the resulting interpolation, although the reasons are different for the piecewise linear and the natural neighbour interpolation. This led to the decision to abandon rather time-consuming efforts to improve the clipping technique (actually, it would have to be fully redesigned).

Instead a simple pre-processing similar to that used for patch-based method (see part 4.2) has been used. Before the interpolation on Voronoi diagram, either the linear interpolation or Natural Neighbour interpolation, the image is interpolated using the Bilinear interpolation. Then the interpolation on Voronoi diagram is processed. Those pixels, which are possible to interpolate using this interpolation, are rewritten and the rest remain with the values calculated by bilinear interpolation.

4.4. Natural neighbour interpolation

Although all the problems concerning interpolation on Voronoi diagram and described in part 4.3 apply to all of the interpolations on Voronoi diagram, including Natural Neighbour interpolation, there are yet several other special difficulties in this method.

The main part of the algorithm, i.e. the creation of the new cells, consists of three steps described in part 3.6. As it is rather complicated process, it will be described more thoroughly in the following text. To remind the algorithm:

> *1) Construct a line between the inserted point and the control point of the participating cell.*
> *2) Find the centre of this line and construct a parallel line, passing thru the centre.*
> *3) Find the intersections A and B of the parallel line and edges of the cell. Line AB is an edge of the new cell.*

In the beginning, one of the not-yet processed cells is chosen (and removed) from the list of not-yet processed cells, which are the participating (see part 3.6) cells for the inserted point. Steps one and two of the algorithm are done and then begins the search for intersections (step 3). Starting from the first vertex of the cell and continuing in a counter clockwise order, each edge of the cell is tested against the parallel line computed in step two. If an intersection is found, its location is computed and it is stored.

A singular case of the intersection being vertex of the cell is tested here. If it occurs, two edges are candidates for beings shared with a participating cell (if the intersection is in vertex Q_j, it can be either edge Q_{j-1}-Q_j or Q_j-Q_{j+1}). The closer one to the inserted point is chosen.

Moreover, if such intersection is found, it would be found again while checking another edge of the cell for intersections. Therefore, whole edges are not considered for testing, but only part of them – without one of the terminal vertices (it is always contained in the neighbouring edge, so it is assured that it won't be missed).

To get the stolen area, another two lists of vertices are kept. They represent the two polygons that develop from the participating cell when it gets divided by the edge of the new cell. Until the first intersection is found, all processed vertices of the cell are appended to the first polygon. When the first intersection is found, it is added to the first polygon and also it is set as a first vertex of the second. Then until the second intersection is found, all vertices are appended to the second polygon. When the second intersection is found, it is the last point of the second polygon and also another point of the first. Then rest of the cell is added to the first polygon.

To find out which of these polygons is the stolen area and which is just the rest of the cell, the known orientation of the cell (counter clockwise in the presented implementation), and therefore the polygons as well, can be used. A simple "sign-test" (cross product) of the dividing (new) edge and the inserted point shows which polygon is the right one (is closer to the inserted vertex). The area of this polygon is then computed by the formula showed on Equation 4.1 and set as the area stolen from the processed cell.

$$Area = -(V_1 \times V_2 + V_2 \times V_3 + \ldots + V_{n-1} \times V_n) / 2$$

Equation 4.1: Formula for computing an area of a convex polygon. $V_1 \ldots V_n$ are the vertices of the polygon as vectors, i.e. V = (V.X, V.Y), "x" is a cross product. The vertices must be sorted in a counter clockwise order.

There is another singular case when looking for the intersection and that is when the parallel line is identical with an edge of a cell. However, this case should occur only when the inserted point is identical with a control point of some existing cell. Then no intersections are found, only one cell (the initial) is involved in the process and the stolen area is identical with the whole area of the cell. Simply put, the colour intensity value assigned to the cell is the colour of the control vertex, which is just right in this case. Therefore this singular case actually does not need be taken care of in any special way.

5. Experiments settings

5.1. Image measurement methods

Mean square error, abbreviated as MSE, is one of the simplest methods for measurement of image compression quality. Its value is gained as an average of the quadrates of differences in pixels of the original image and of the compressed one. Obviously, lower value means better quality, with zero meaning equal images. Equation 5.1 shows the mathematical formula that corresponds to that idea:

$$a) \quad MSE = \frac{1}{m \cdot n} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} \left[I(x,y) - K(x,y) \right]^2$$

$$b) \quad MSE = \frac{1}{3 \cdot m \cdot n} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} \sum_{k=0}^{2} \left[I_k(x,y) - K_k(x,y) \right]^2$$

Equation 5.1: The formula of the MSE. m *and* n *is the size of the image in pixels,* I(x,y) *is a pixel of the original image in position (x,y),* K(x,y) *is a pixel of the interpolated image. Formula* a *is for greyscale images, formula* b *for coloured images. In the second case,* I_k *and* K_k *are images with only one of the colour components.*

Though widely used, the mean square error actually is not a good tool for measurement of image quality. Its main flaw is that because of the quadrate of pixel differences, a huge weigh is assigned to outliers, i.e. pixels, for which the difference is very big. Though this might seem reasonable, imagine the following scenario: there is an image, exactly the same as the original one, only with few (e.g. less then one percent) scattered pixels having absolutely black or white, whichever is farther from the original. Such image will be rated very poorly by MSE, while human's eye might not even notice the flawed pixels.

More common method used for image comparison is the "peak signal to noise ratio", or PSNR. Equation 5.2 shows how it is computed.

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX^2}{MSE} \right) = 20 \cdot \log_{10}\left(\frac{MAX}{\sqrt{MSE}} \right)$$

Equation 5.2: The formula of the PSNR. MSE *is the mean square error of the image,* MAX *is the maximal possible value in a pixel, usually 255 (for system with 8-bits per colour channel).*

Unlike the MSE, PSNR is measured in dB, the more, the better. Usual values are around 20 – 40 dB, where 30 dB is usually stated as a threshold between the ability to recognize the compressed image from the original. Although, as can be seen in the equation, the PSNR is still based on the MSE, and therefore it does not actually solves the problem of low correlation between it and human perception, i.e. images rated as better then others by PSNR often get opposite rating from humans, and vice versa.

Structural similarity index, or SSIM, is a relatively new metric, which in most cases overcomes the PSNR when compared with human's eye rating, while having equal computation power needs. In [9] is described a way how to numerically express structural

information about the image, which is what a human is used to extract from the image. Therefore the SSIM should be more suitable for image comparisons, as it more correlate with the human perception system.

Unlike the previous methods, SSIM operates with larger structures then one pixel, e.g. 8x8 (or other size) pixels windows. This helps to prevent the already mentioned problem with outliers.

Equation 5.3 shows the formula of how to compute the index.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)\cdot(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)\cdot(\sigma_x^2 + \sigma_y^2 + C_2)}$$

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i; \quad \sigma_x = \sqrt{\left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)}; \quad \sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}\left[(x_i - \mu_x)\cdot(y_i - \mu_y)\right]$$

$$C_1 = (K_1 \cdot MAX)^2; \quad C_2 = (K_2 \cdot MAX)^2$$

Equation 5.3: The formula of the SSIM (with additional declarative formulas). x *and* y *are the pixel windows, μ is an average of intensity in given window, σ_x and σ_y are the square roots of variance, σ_{xy} is the covariance, C_1 and C_2 are constants, which avoid instability when ($\mu_x^2 + \mu_y^2$) or ($\sigma_x^2 + \sigma_y^2$) are close to zero.* MAX *is the maximal possible value in a pixel (usually 255),* K_1 *and* K_2 *are small (much smaller then one) constants, N is the size of the window.*

To get one rating for the whole image, a mean structural similarity index (MSSIM) can be used. It is computed as an average of all the SSIM of the image. SSIM takes values between -1 and 1, with 1 meaning equality of images.

Although the MSSIM usually behave better then MSE or PSNR in terms of being close to rating by human perception, the authors of [9] state that it tends to be dependent on the input. Therefore the images presented in chapter 6 will be compared by both MSSIM and the standard methods, i.e. MSE and PSNR.

5.2. Colour systems

For the purpose of coloured images, two colour systems were considered. First of them is the RGB system, simply because it is undoubtedly the most commonly used system when it comes to digital image. The second system is the YCbCr. This system was regarded as the best for the purpose of triangulation represented digital image in [8]. As the topic of this thesis is closely related to [8], it seemed more then reasonable to take its conclusions into account.

The most common way to represent colours in today's computers is to use 32-bit number, where first 8 bits are reserved for the alpha channel[6] and the rest are used for the individual

[6] The transparency of the image, where maximal value means solid image and zero means fully transparent.

components of the respective colour system, 8-bits each. This implies the maximal and minimal value ranging from 0 to 255.

The RGB is an additive colour system, which means that the intensity is added to the intensity of the background, resulting in more intensive colours the bigger the value of the RGB components are, with white being the maximal value and black being the minimal (zero) value. This makes additive systems like RGB suitable for monitors and other display hardware, because they emit light, so they correlate well with the system – the bigger the value, the more light is emitted and vice versa.

To create a coloured image, one light emitter would not suffice. Therefore a blending of more basic colours is used generally. In case of RGB, these basic colours are, as the name suggests, red, green and blue.

The RGB system has proven to have too much of redundant information to be used for transmission. This led to development of other system, YCbCr, which would be able to store almost the same information as the RGB colour but with less bit per pixel demand. The way to do this was to separate the intensity value (Y component) from the chromatic blue (Cb) and red (Cr) modifiers, which are actually differences between the intensity and the blue or red component. While the Y component still needs to be transferred in high resolution, the Cb and Cr can be reduced and compressed without influencing the overall quality of the resulting image.

5.3. Testing subjects and environment

All the results presented and discussed in chapter six were created using the “Interpolator” software, which is a part of this thesis. All the experiments were processed on a PC station with the following configuration:

- Processor: Intel Pentium Core2Duo E6300 (two cores, processor clock 1.86GHz per core, 2MB L2 shared cache)
- Memory: DDR2 Kingmax PC800 1GB (memory clock 800Hz)
- Operating system: Windows XP SP2

10 greyscale and 7 coloured images were used as the testing subjects. These images were picked in order to incorporate the commonly used images (like the Lena or Baboon images), classical scenery images (the Yacht), images with sharp colour edges (the Peppers, the Baboon), images with low colour variety (the Airplane, the Maran) as well as with tiny details (the Boat, the Lukas) and also one rendered image, the Pool.

6. Experiments

6.1. Quality comparisons

The three following parts will present the results of individual interpolation methods when applied on the set of images described in part 5.3 and compare each to other. The first part (6.1.1) focus on greyscale images, the second part (6.1.2) describes the application on coloured images represented by separate triangulations and the third part (6.1.3) shows the results gained when using the co-triangulations.

The interpolation methods themselves do not change at all no matter if the image is coloured or grey. Therefore their performance does not really differ when used on the coloured image. For this reason, there are more data and statistics in the first part (6.1.1), as the typical characteristics and behaviour of individual methods is more clear and easier to see on the case with only one triangulation per image. The other two parts then describes only differences between the greyscale and coloured images, i.e. the impact of the colour systems described in part 5.2, relation between vertex count of individual triangulations and the quality of resulting image (as more information is obviously needed to interpolate a coloured image then a greyscale one), etc.

The data presented in the following text is usually only one typical image, which represent most of the other tested images in terms of achieved quality (using quality measurement methods described in part 5.1). On the opposite side, images that were atypical in some way are also presented and discussed. However, if you are interested in seeing more experiment results anyway, please refer to the material appended to this thesis.

6.1.1. Greyscale images

Greyscale images were tested with five different vertex counts per triangulation – 1000, 2000, 5000, 10000 and 15000. Naturally, the more vertices there are in a triangulation, the better is the result. The images (both original and interpolated) are in 512x512 pixel resolution, with these exceptions: the Maran image (400x300), the Kodim image (465x375) and the Monarch (768x512). First set of charts (figures 6.1 – 6.3) compares the quality of the images interpolated by the individual methods.

Figure 6.1 present charts of the Lena image. Most of the other tested images have similar results (see the appendix for charts of more images), with the Bilinear and Zienkiewicz's interpolation leading independently on the measurement methods. On the other side, the piecewise linear interpolation on Voronoi diagram fails significantly. Either way, the SSIM ranging from 0.5 to 0.7 is a sign of a poor similarity between the interpolated and original images. This actually is not very surprising, 1000 vertices for a 512 per 512 pixel image means 97.38 % of the initial colour information has been lost.

The absolute ratings of individual images vary, dependently on how many tiny details and empty spaces there are in the image. For example, Figure 6.2 show comparison charts for the Maran image, which has flat, empty background and only the centre of the image is covered. Such image gets good rating, with SSIM rising over 0.8. However, as can be seen on Figure 6.3, the resulting image does not look any better for humans then the other images, because a human will still see a blurry object in the centre. That the simple background looks exactly the same as on the original is not important for most people, even if it covers most of the image.

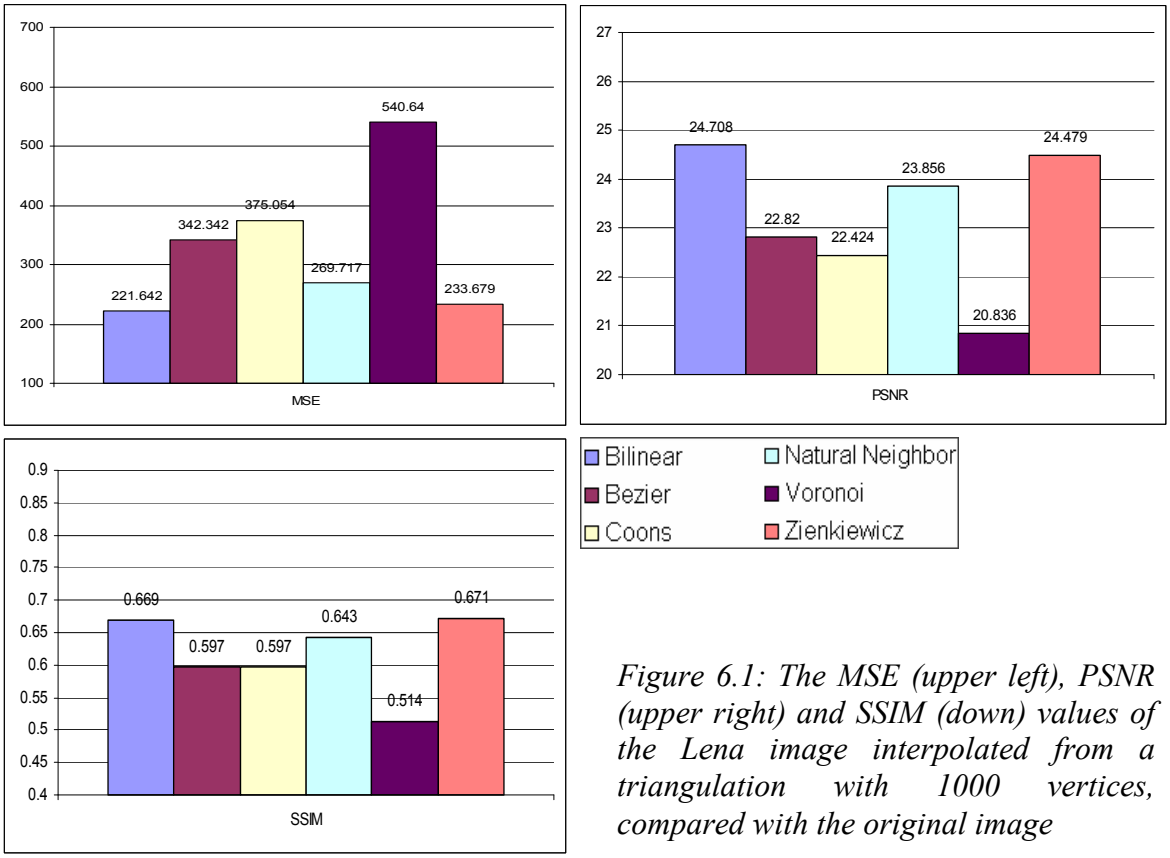

Figure 6.1: The MSE (upper left), PSNR (upper right) and SSIM (down) values of the Lena image interpolated from a triangulation with 1000 vertices, compared with the original image

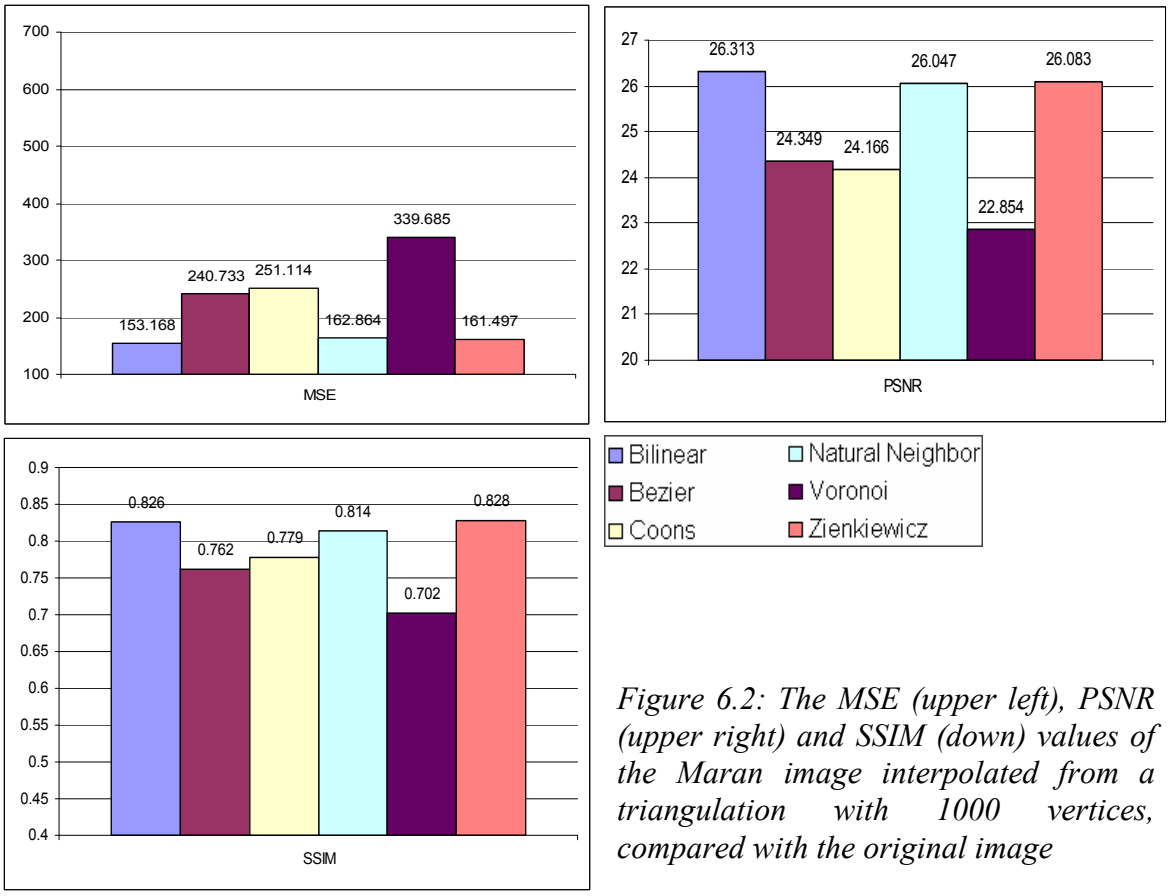

Figure 6.2: The MSE (upper left), PSNR (upper right) and SSIM (down) values of the Maran image interpolated from a triangulation with 1000 vertices, compared with the original image

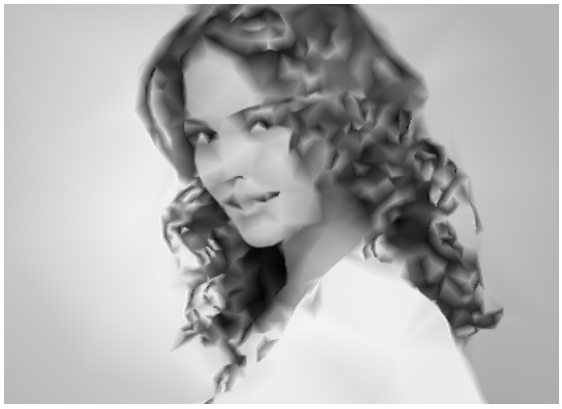

Figure 6.3: The Maran image, bilinearly interpolated.

Increasing the vertex count to 2000 does not bring any significant changes to the performance of individual methods. There is, however, one interesting thing. The charts of the Lukas image, Figure 6.4, show unexpectedly large MSE (and therefore small PSNR) for the Coons patch method. This is due to artifacts, to which this method is vulnerable.

The reason of these artifacts is more evident with increasing vertex count. Figure 6.5 show images of the 10000-vertex triangulation of the Lukas image as well as the result of the Coons interpolation on this triangulation. The artifacts appear when triangles of very different sizes meet in one patch. As was described in part 3.4, the boundary curve of the coons patch is constructed in such a way, that the middle vertex of the curve is a peak of the curve. When two very different triangles meet, the spatial distances between the peak vertex and the terminal vertices will be also very different. The impact is that as these points are forced to create one continuous curve, this curves shape will be very distant from the shape of the polyline created by the same vertices as the curve.

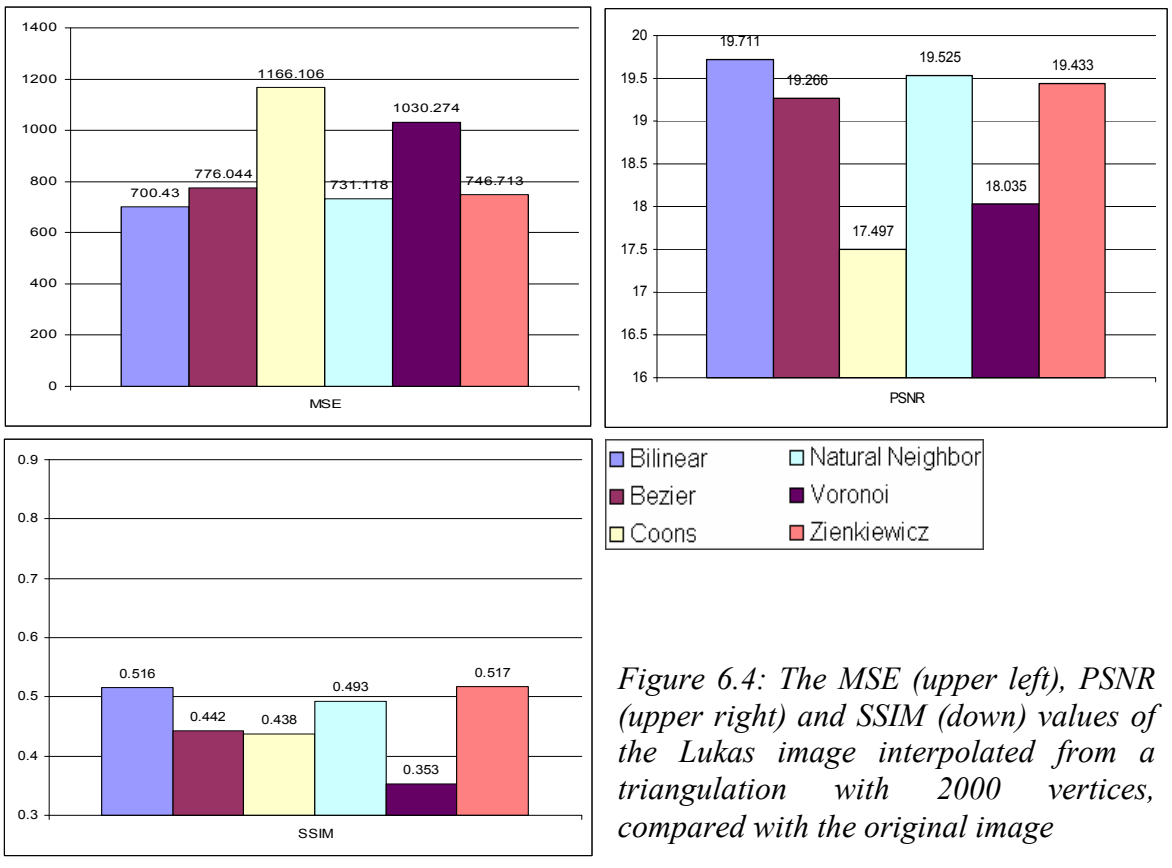

Figure 6.4: The MSE (upper left), PSNR (upper right) and SSIM (down) values of the Lukas image interpolated from a triangulation with 2000 vertices, compared with the original image

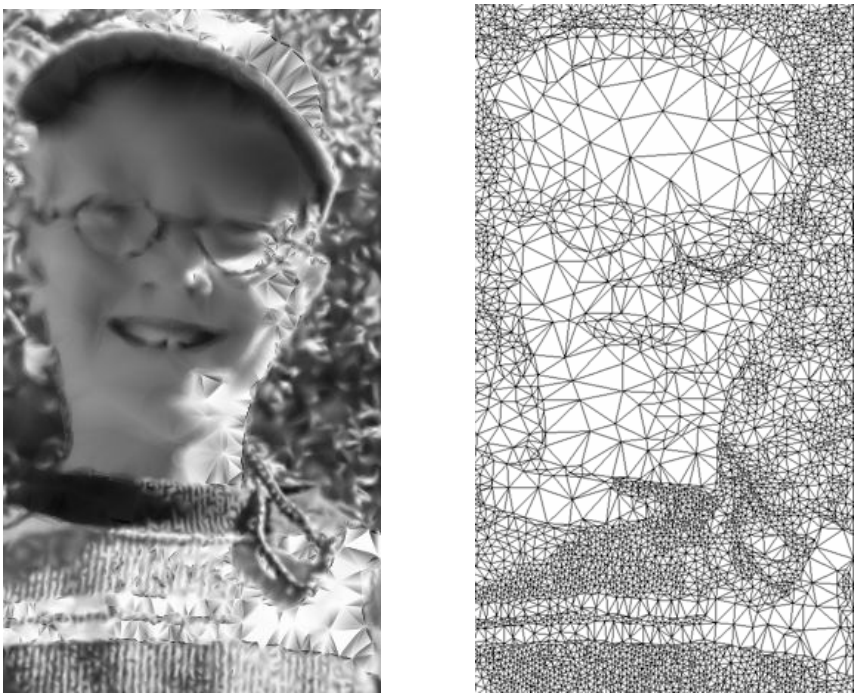

Figure 6.5: Part of the Lukas image triangulation with 10000 vertices (right), and its interpolation by the Coons patch method (left).

It is quite interesting that these artifacts do not seem to influence the SSIM much. On the other hand, the piecewise linear interpolation on Voronoi diagram fails in every possible measurement method, including human eye perception, even though it does not suffer from similar singularities.

With 5000 vertices per triangulation, the interpolated images are getting much closer to looking like the original ones. The Bilinear, Zienkiewicz's and Natural Neighbour interpolations are proving to be the best among the set. Figures 6.7 and 6.8 show comparison charts for the Lena and Peppers images. Note, that the results are almost the same for both images, with one difference. The PSNR of Bilinear and Zienkiewicz's interpolation is significantly bigger for the Peppers image. This is suggested to be a matter of sharp edges in the image. The Peppers image contains a lot more of them, which has its impact on the created triangulation. Figure 6.6 shows that this results in a strip of tiny triangles being along these sharp edges.

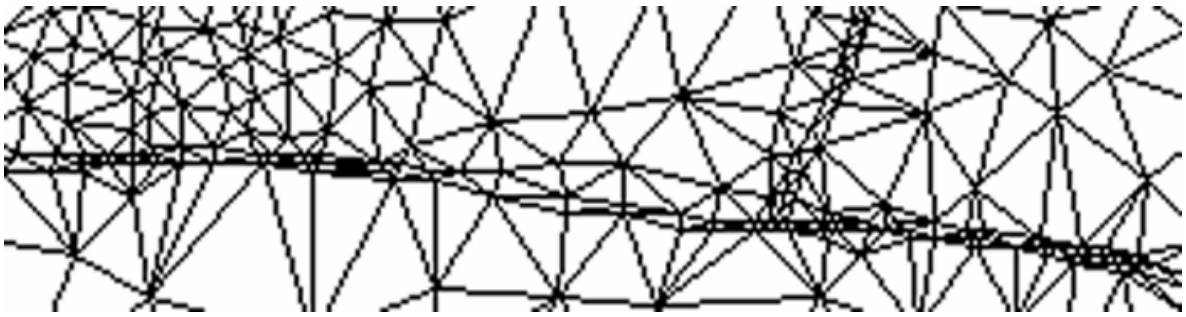

Figure 6.6: A small portion of the Peppers triangulation with 10000 vertices. Note the strips of very small triangles along colour edges.

This evidently helps the Bilinear and Zienkiewicz's interpolation to preserve edges better. On the other hand, a drop in performance of the Coons patch interpolation can also be seen, which is due to the artifacts mentioned before – the tiny triangles meet with the large ones inside individual peppers, which are large smooth areas and therefore represented by big triangles. The Bezier patch interpolation is also affected by similar artifacts as the Coons patch interpolation, also with similar reasons. However, as the charts show, their impact is not as huge as in case of the Coons patch.

One look on Figure 6.9 is enough to see, that the piecewise linear interpolation on Voronoi diagram is simply useless. One could assume from the charts that the same is true for the patch-based methods. However, closer look on the results of their performance, as can be seen on Figure 6.10 shows, that these methods have their merits.

The images on that figure are the results of the two patch-based methods and the Zienkiwcz's method for comparison. As expected, the colour edges are blurred. But the flat areas are smoothed perfectly, leaving no trace of the underlying triangulation – which is something that cannot be said about the overall favourites Bilinear and Zienkiewicz's interpolation.

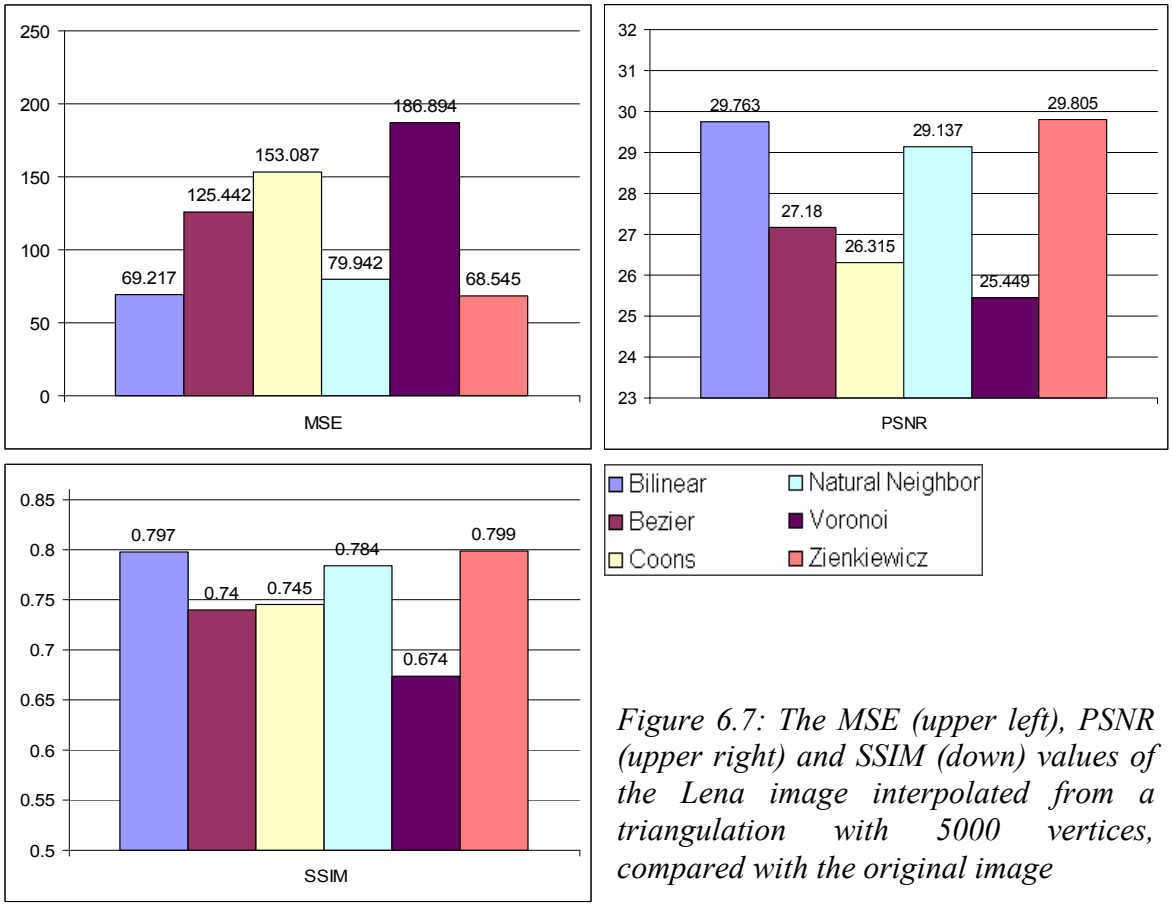

Figure 6.7: The MSE (upper left), PSNR (upper right) and SSIM (down) values of the Lena image interpolated from a triangulation with 5000 vertices, compared with the original image

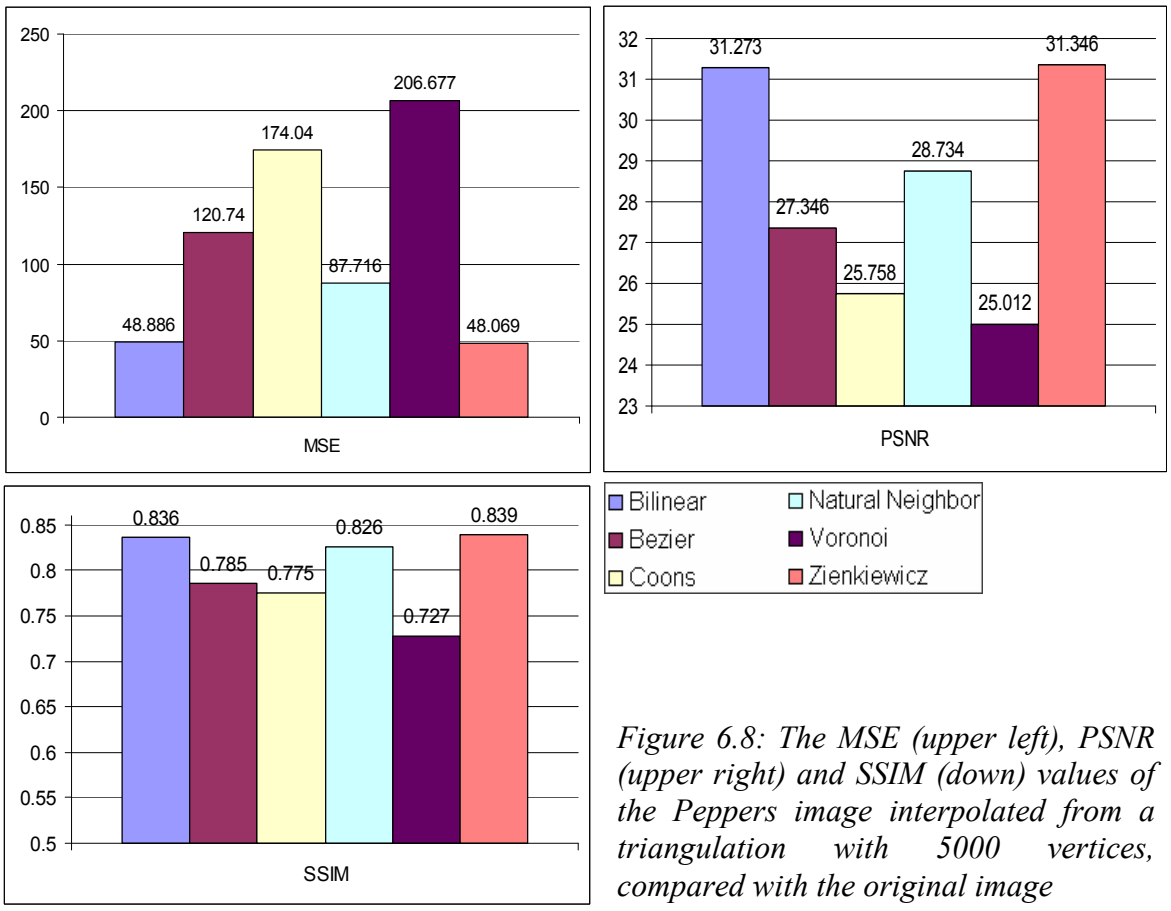

Figure 6.8: The MSE (upper left), PSNR (upper right) and SSIM (down) values of the Peppers image interpolated from a triangulation with 5000 vertices, compared with the original image

Figure 6.9: Part of the Fruits image, as interpolated by the piecewise linear interpolation on Voronoi diagram from a 5000-vertex triangulation.

Figure 6.10: Part of the Lena image interpolated from a 5000-vertex triangulation by Bezier patch (upper left), Coons patch (upper right) and Zienkiewicz's (down) interpolation.

With vertex count increasing to 10000 and 15000, nothing surprising happens. The Zienkiewicz's interpolation outperforms the Bilinear interpolation (as well as any other) on every tested image, securing its position as the "best" method.

The Natural Neighbour interpolation is able to keep pace with the Zienkiewicz's and Bilinear interpolations. Acknowledging the problem with interpolating the border of the image with that interpolation (see part 4.3 for details), which certainly has a negative effect on the results, this interpolation method seems quite perspective.

With 10000 vertices the PSNR is usually going over 30 dB, which is commonly considered as a limit for being able to recognize the image from the original. Although several images with a lot of details, like the Boat, can still be easily recognizable from the original image, on the "average" images, the flaws of the interpolations are only minor (see the appended material for your own comparisons).

Though the quality of the results of patch-based methods improve with the vertex count, it is clear that no matter how many vertices there are, these methods cannot be used universally, because there will always be some artifacts present. However, there might be a

way of adjusting the construction of the triangulations in such a way that could make these methods useful, at least for interpolation of some, the "smooth", parts of the triangulation.

Lastly, on Figure 6.11 is a chart, which describes the effect vertex count has on the quality of the interpolation for individual methods. The Fruits image was chosen for this measurement, because it contains some small details and sharp edges as well as some flat areas. It documents, that the differences between the 10000 and 15000 vertices are small, more or less insignificant. Note that the 30 dB limit in PSNR can is reached already with 5000 vertices.

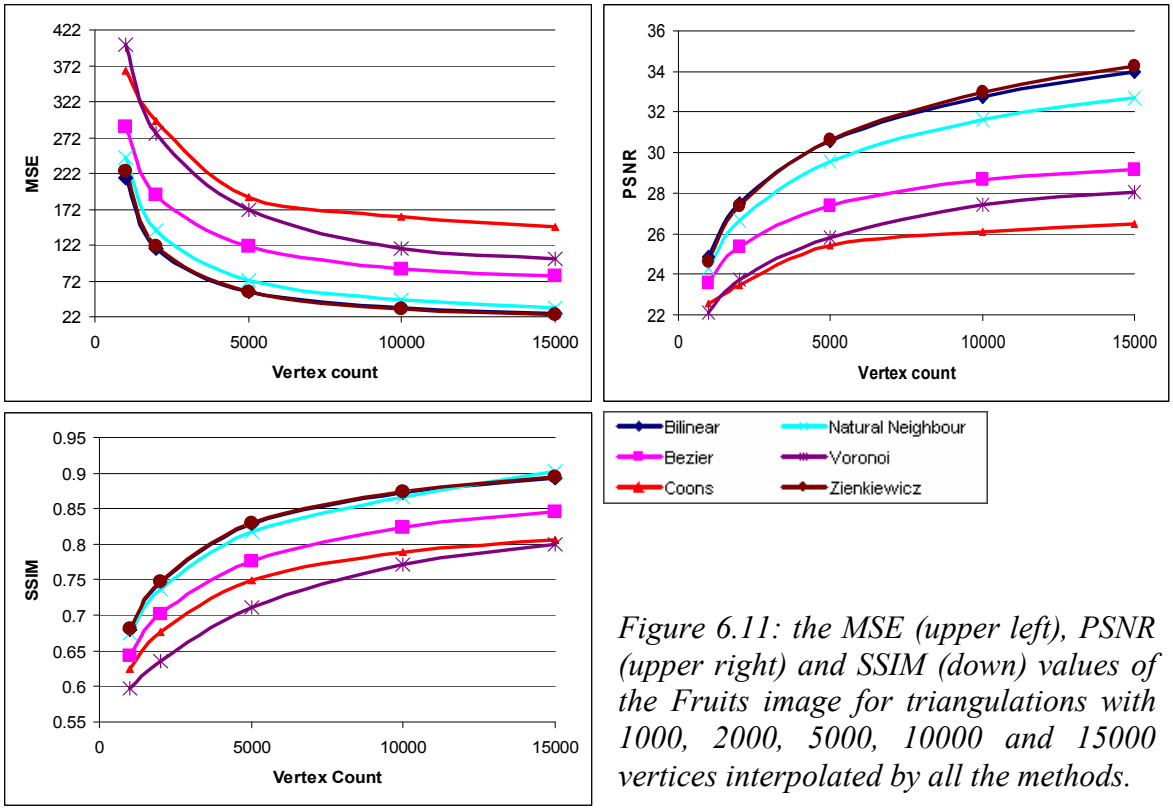

Figure 6.11: the MSE (upper left), PSNR (upper right) and SSIM (down) values of the Fruits image for triangulations with 1000, 2000, 5000, 10000 and 15000 vertices interpolated by all the methods.

6.1.2. Separately interpolated coloured images

This part will analyze the performance of the individual methods when applied to coloured images represented by three separate triangulations, one for each colour component. Unlike with the Greyscale images, the triplets of triangulations were not chosen based solely on their vertex count, but rather on the PSNR they achieve when interpolated by the Bilinear interpolation when compared to the given component, extracted from the original image.

For the RGB colour system, this meant simply finding a "green" and a "blue" triangulation, which PSNR is nearest to a PSNR of a given "red" triangulation. As the Cb and the Cr components of YCbCr system contain less information then the Y component, they are more likely to look like the original components extracted from the image. Therefore the Y component obviously scored much lower then the other two. To be able to put different triplets of triangulations together, a constant value was added to the PSNR of the Y component to make it more or less matching with the interval in which the PSNR values of Cb and Cr components usually were.

The triangulations were chosen from a set of 20 triangulations for each colour component, ranging from 1000 to 20000 vertices. In the following text, results for some of these triplets will be shown and discussed. All the images are in 512x512 pixel resolution, with one exception, the Pool image (510x383).

In general, the performance of individual methods when interpolating coloured images is similar to the greyscale case. As charts on Figure 6.12 document, the Zienkiewicz's and Bilinear interpolation are still the best, followed closely by the Natural Neighbour interpolation. The data on these charts are from the Lena image, interpolated from various triplets of Triangulations, using the RGB colour system.

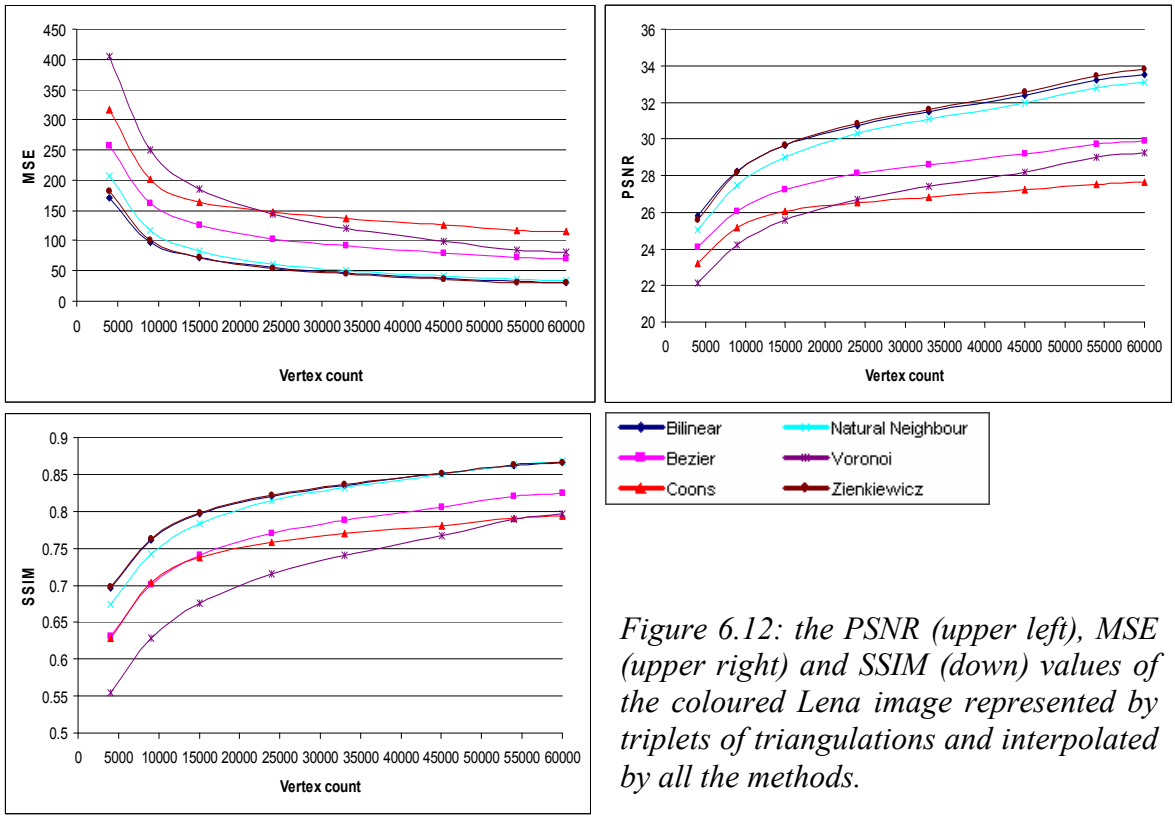

Figure 6.12: the PSNR (upper left), MSE (upper right) and SSIM (down) values of the coloured Lena image represented by triplets of triangulations and interpolated by all the methods.

The numbers of vertices presented on the charts are a sum of vertices in all the three triangulations. The vertex count in individual triangulations is more or less equal simply to the third of this sum – in general, the components are equal in terms of how much information they carry, although in some pictures some colour component can be, and usually is, dominant.

In the case of the YCbCr system, things get more complicated, as the Y component carries much more information then the remaining components. After all, this is why the system was designed in first place. This brings some problems into the evaluation process of the quality of the image. While the number of vertices in the triangulation, which represents the Y component, influences the resulting image greatly, the impact of the Cb and Cr components is much smaller.

Figure 6.13 presents comparison charts for an image represented by YCbCr triangulations. There are two cases, one using a triangulation of the Y component with 3000 vertices and with 7000 vertices. The triangulations of the Cb and Cr components added to these

triangulations have 1000 vertices each in one case and 20000 vertices in the other one. I.e., there are four test cases – one with 3000 vertices in Y and 1000 in Cb and Cr, then 3000 vertices in Y and 20000 in Cb and Cr, 7000 vertices in Y and 20000 in the other components and lastly 7000 in Y plus 20000 in each of the others. The difference between 1000 and 20000 vertices per Cr and Cb triangulations has only a minor effect to the measurement statistics, while on the other hand, raising the vertex count by "only" 4000 vertices in the Y component matters much more.

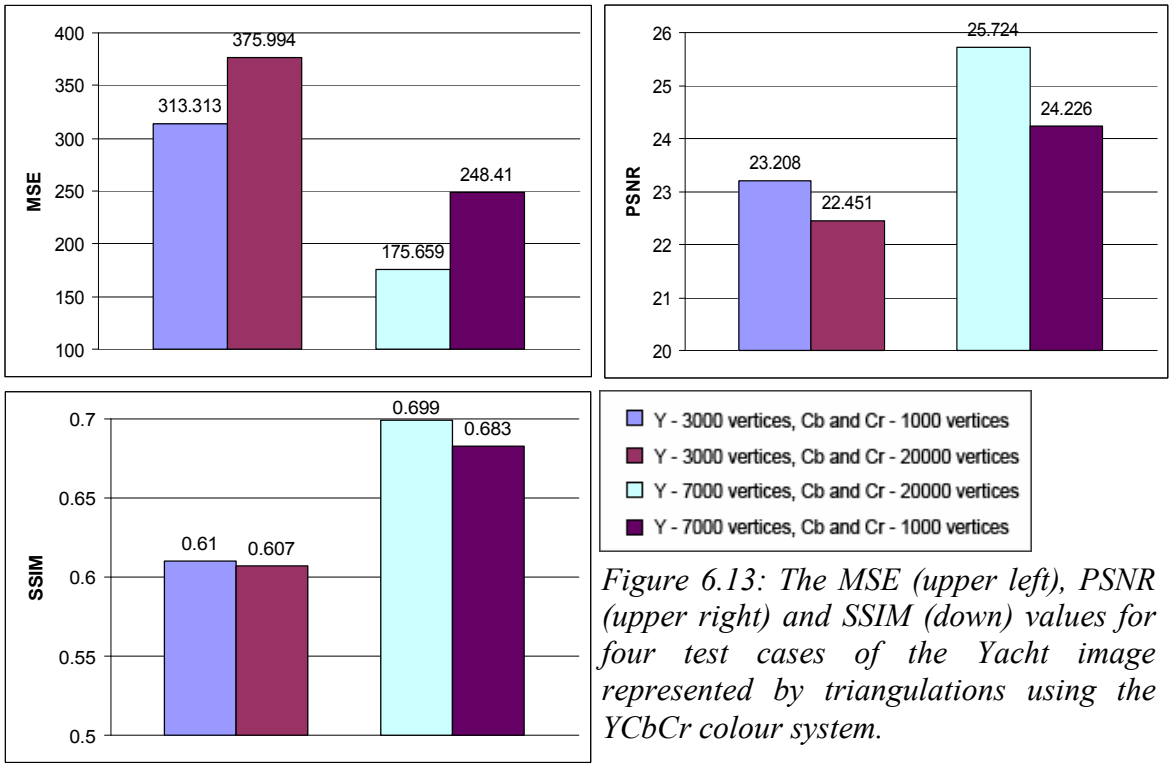

Figure 6.13: The MSE (upper left), PSNR (upper right) and SSIM (down) values for four test cases of the Yacht image represented by triangulations using the YCbCr colour system.

This could lead to a conclusion (and as [8] proves, a right one) that the YCbCr might be more useful when better compression of the image is a priority, because most of the "important" information about the colour can be stored by only one component instead of three. Figure 6.14 uses the Yacht image, a typical scenery photo, to compare the impact the two colour systems have on the quality of the interpolated image. As the charts show, the differences are rather minor, but still the RGB is slightly better. Therefore it seems like a good choice when the quality of the image is preferred before compression.

On the few following figures, more atypical cases will be presented. On Figure 6.15, there is a comparison for the Baboon image. This image has a lot of colour edges and therefore it is not a surprise, that even with relatively large vertex count, the results are not very good, independently on the used interpolation method.

As in the case of greyscale images, the artifacts produced by patch-based methods are obviously present in the coloured images as well. Moreover, the distortion in colour hues is much more distracting for human eye then the distortion of the greyscale intensity. Therefore even when the effect on the measurement statistics is the same as with greyscale images, artifacts in coloured images are less likely to be overlooked by a human.

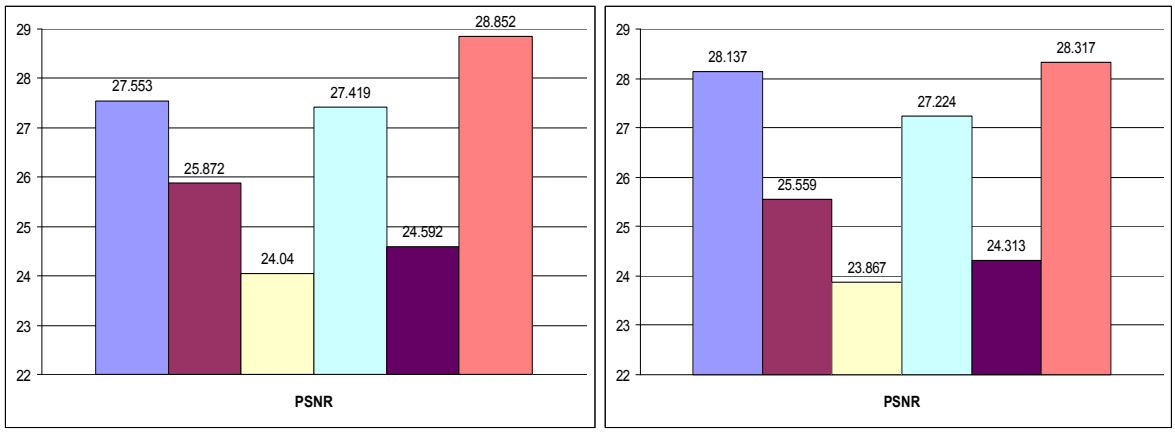

Figure 6.14: A comparison of two of the Yacht images, interpolated from triangulations with 20000 vertices in each component. Image in RGB system is on the left, YCbCr on the right.

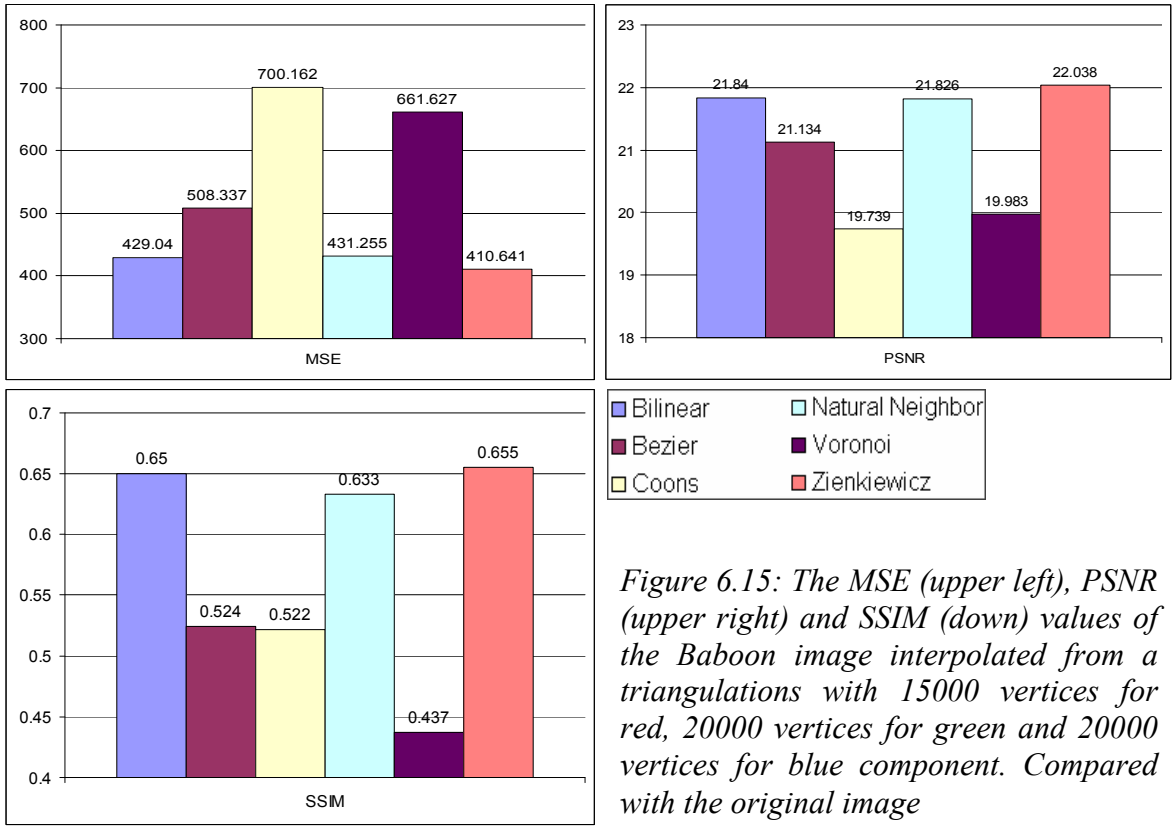

Figure 6.15: The MSE (upper left), PSNR (upper right) and SSIM (down) values of the Baboon image interpolated from a triangulations with 15000 vertices for red, 20000 vertices for green and 20000 vertices for blue component. Compared with the original image

Figure 6.16: Part of the Fruits image, interpolated by the Coons patch method from triangulations with 6000 vertices in red, 9000 vertices in green and 9000 vertices in blue component.

Figure 6.16 shows part of the Fruits image, which's red component suffers from artifacts. That results in rapid change of hue on the afflicted parts. Note that even though these artifacts are not present in the YCbCr version of the image, it is solely a matter of the geometry of the triangulation. I.e. there might be other image, which would be perfect in RGB system, but produced artifacts in the YCbCr system.

6.1.3. Coloured images represented by Co-triangulation

The following text evaluates the differences between coloured images represented by separate triangulations and by co-triangulations. Figures 6.17 and 6.18 shows comparison charts for the Yacht and the Baboon images when interpolated from separate triangulations and co-triangulations. The images behaved similarly for both tested colour systems, therefore the results presented on Figures 6.17 – 6.20 all use the RGB colour system.

An interesting finding is that while the PSNR value is better for separate triangulations, while the SSIM is better for co-triangulations. This behaviour is quite expectable. The value of PSNR is an average of PSNR for each colour component. As the separate triangulations are constructed from (and compared to) individual colour components, they are likely to be similar to them. On the other hand, the co-triangulation is a compromise between individual components. Therefore lower PSNR values are obtained when the components are extracted from the co-triangulation and compared separately. The SSIM is designed to express overall structural similarity of the image. Because the co-triangulation is constructed with respect to the whole image (all its components at once), it is more likely to be able to capture the "structure" of the image then separate triangulation can.

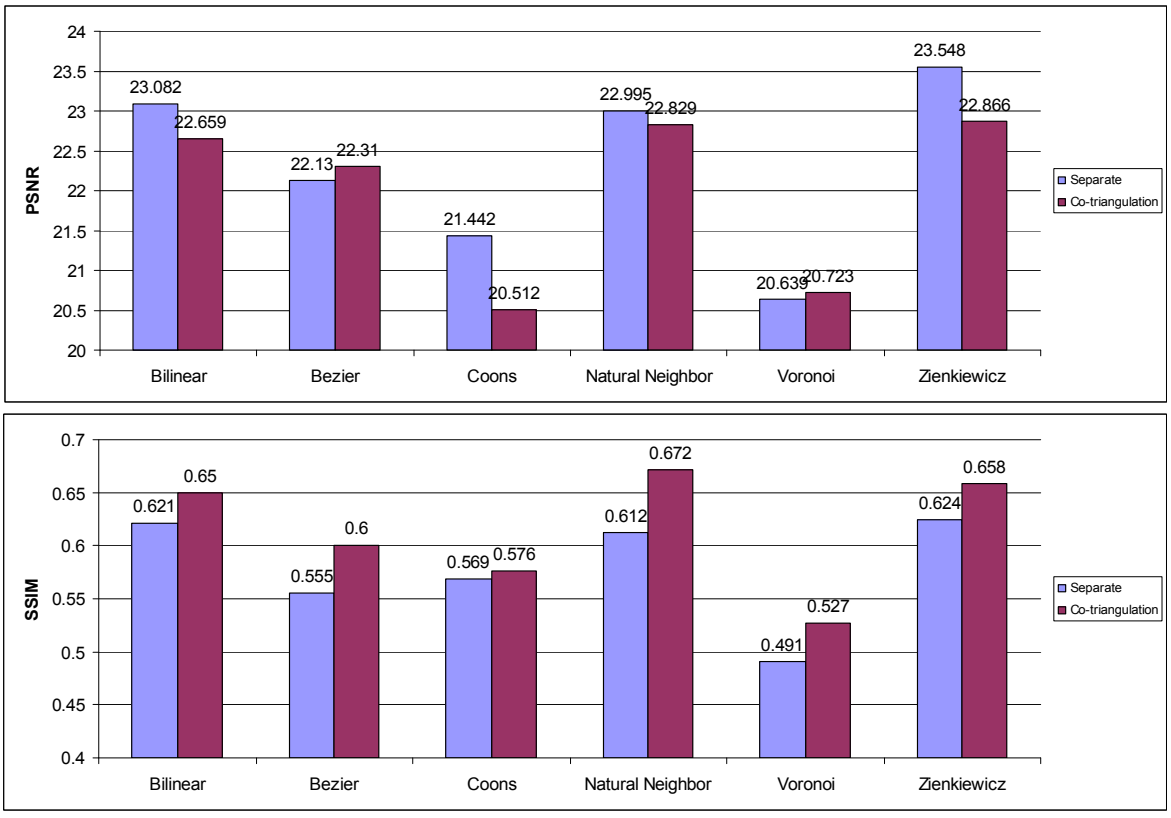

Figure 6.17: The PSNR (upper) and SSIM (down) values for the Yacht image interpolated from separate triangulations (blue colour) with 10000 vertices (1000 red, 4000 blue and 5000 green component) and from co-triangulation (red colour) with 7000 vertices.

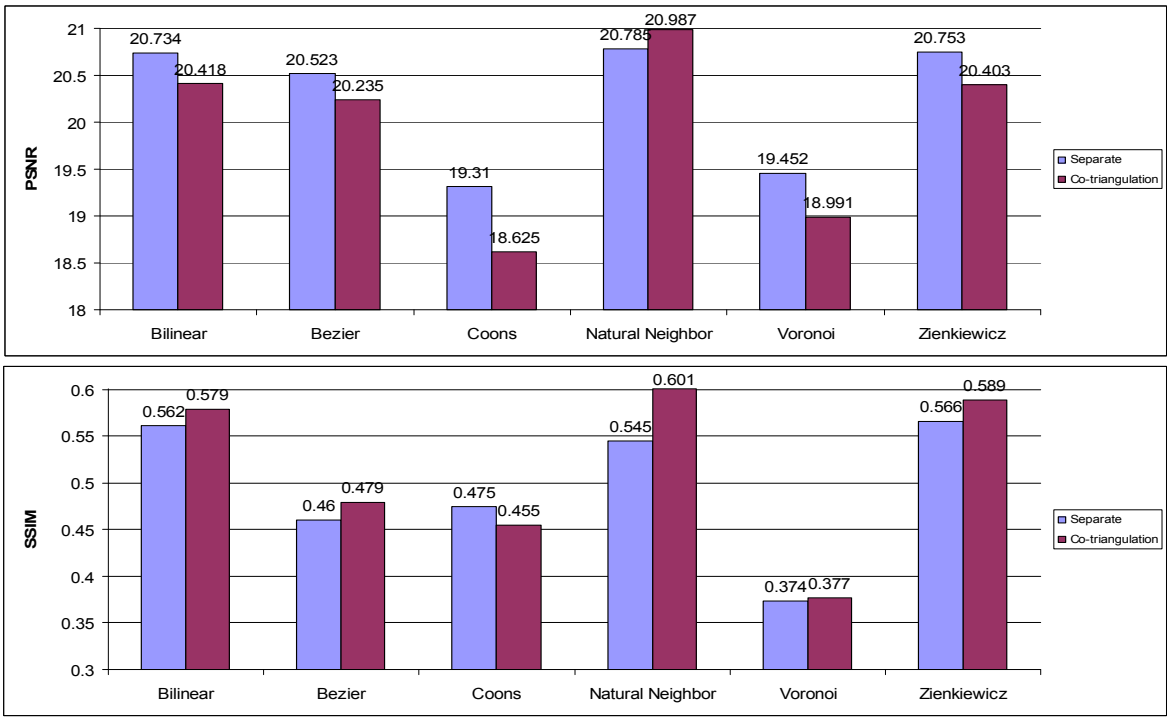

Figure 6.18: The PSNR (upper) and SSIM (down) values for the Baboon image interpolated from separate triangulations (blue) with 35000 vertices (7000 red, 14000 blue and 14000 green component) and from co-triangulation (red) with 19000 vertices.

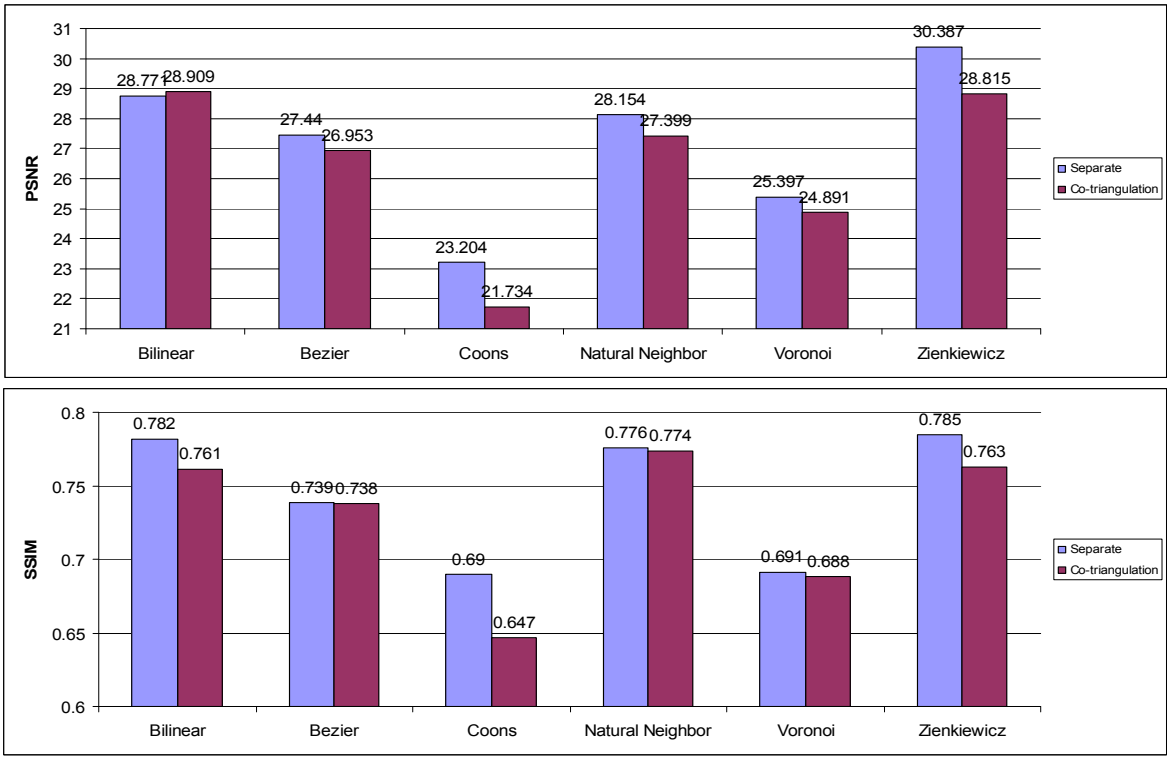

Figure 6.19: The PSNR (upper) and SSIM (down) values for the Peppers image interpolated from separate triangulations (blue colour) with 20000 vertices (7000 red, 6000 blue and 7000 green component) and from co-triangulation (red colour) with 10000 vertices.

On the other hand, Figure 6.19 and 6.20, showing charts of the Lena and the Peppers image, prove that this is not always true, as the comparison statistics behave as usual, i.e. the better the PSNR, the better the SSIM. As there is no obvious resemblance between the above mentioned two pairs of images, whether in what they depict or how their triangulations look like, it is hard to tell which of these cases are atypical in terms of "separate triangulations versus co-triangulations" behaviour. Much more tests (tens to hundreds of images) would have to be made in order to find out. As the co-triangulations are only a minor subtopic of this thesis and taking into account the time demands of such extensive testing, this problem is to remain unsolved for now.

The main merit of the co-triangulation is its effect on the compression ratio. Note that images depicted on Figures 6.17 – 6.20 needed fewer vertices for co-triangulation to achieve similar PSNR and SSIM scores as images interpolated from separate triangulations with significantly more vertices.

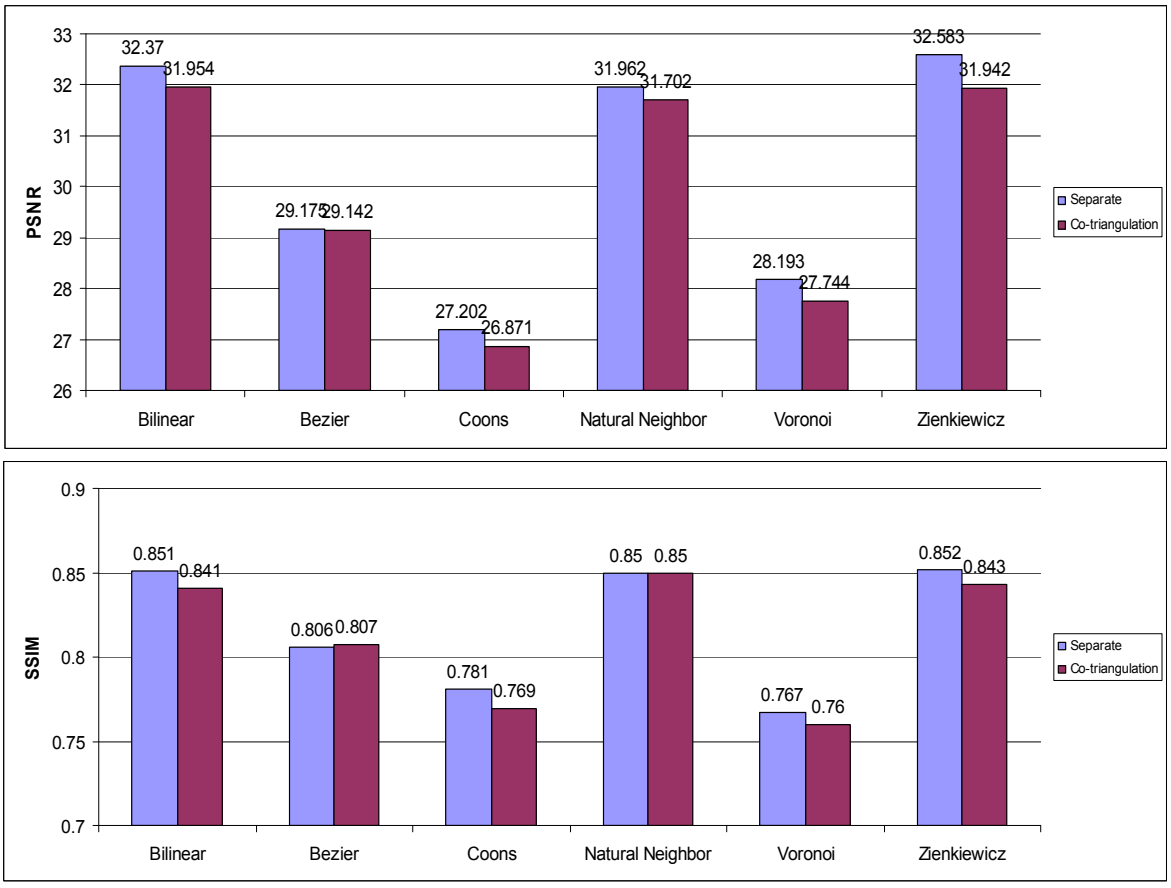

Figure 6.20: The PSNR (upper) and SSIM (down) values for the Lena image interpolated from separate triangulations (blue colour) with 45000 vertices (7000 red, 20000 blue and 18000 green component) and from co-triangulation (red colour) with 19000 vertices.

Automatic measurement methods put aside, images interpolated from co-triangulations have one flaw, which is usually easy to spot for a human. On Figure 6.21, two images with very similar PSNR and SSIM values are compared, one interpolated from co-triangulation, one from separate triangulations. Traces of the triangular structure can be seen on some parts of the image interpolated from co-triangulation, while the other one is smooth in those parts. This has quite obvious reason, as the separate triangulations does not (usually)

have the same structure. Therefore any visible transition between individual triangles, which might occur during the interpolation, is in most cases overlaid when the three components are blended together. Because the structure is the same in for all colour components in the case of co-triangulations, any visible transitions remain.

Figure 6.21: A part of the Lena image interpolated by Zienkiewicz's method from separate triangulations (upper) with 60000 vertices (20000 for each component) and from co-triangulation (down) with 32000 vertices. The upper image has PSNR 33.778 and SSIM 0.866, the lower has PSNR 33.858 and SSIM 0.864. Note the triangle traces in the face, on the nose and in the surrounding area on the left and on the right, which makes the lower image looks more "rugged", even though it has better PSNR and almost the same SSIM.

6.2. Time comparisons

This part evaluates the time demands of individual interpolation methods. The time complexity of the methods is completely independent on the used colour system or the triangulation geometry. The only important parameter is the number of triangles in the triangulation.

Moreover, the implementation, which is part of this thesis, assumes that each triangulation is fully coloured, i.e. all the vertices have three different colour components assigned. However, this is true only for co-triangulations. Separate coloured triangulations and

greyscale triangulations have only one colour component per vertex or putted in other way, they have the same value in all three components. The consumed time (will be referred to simply as "the time" in the following text) could therefore be reduced slightly for these cases, because only one instead of three interpolation expressions per pixel would have to be computed.

The time of the patch-based methods is greatly influenced by choice of the "step" (see parts 3.3 and 3.4 for explanation). The step used for all the experiments presented in this thesis was 0.006 for the Bezier patch method and 0.007 for the Coons patch method. These values were chosen experimentally, in order to ensure full filling of even big triangles, i.e. triangles in triangulations with only 1000 vertices for a 512x512 pixel image. Several triangulations were tried in the experiments to make sure the values are right.

As was said before, the only parameter, which influence the consumed time, is the triangle count. However, the parameter used for triangulation construction is the vertex count. Therefore it is more useful to find what is an average consumed time based on the vertex count, because that is a parameter that can be modified – the number of triangles for a given vertex count can be various, though still within some bound (see part 2.2).

Figure 6.22 shows a table of times for several vertex counts. The statistics were obtained as an average of ten test cases for each vertex count. Note that unlike other algorithms, which are linear in respect to triangle count, patch-based algorithms are quadratic. Therefore their times for individual images may vary greatly from the presented average. The triangulations were chosen randomly only by the vertex count, but triangulations of Cb and Cr components were left out, as they tend to have very small triangle count (they do not carry too much information, see part 6.1.2) as well as some other triangulations with unusual triangle count.

Method / Vertex count	Bilinear	Bezier	Coons	Natural Neighbour	Voronoi	Zienkiewicz
2000	0.0391 s	4.8888 s	17.2735 s	3.0807 s	0.6095 s	0.0656 s
4000	0.0531 s	9.5922 s	33.8125 s	3.2422 s	0.6531 s	0.0813 s
5000	0.0610 s	12.091 s	43.4367 s	3.3354 s	0.6502 s	0.0891 s
7000	0.0688 s	16.75 s	56.6 s	3.4234 s	0.6656 s	0.1141 s
10000	0.0906 s	24.070 s	83.2392 s	3.5322 s	0.6830 s	0.1344 s
12000	0.0938 s	29.064 s	98.2891 s	3.5922 s	0.6938 s	0.1594 s
15000	0.1313 s	37.597 s	128.57 s	3.6463 s	0.7127 s	0.1860 s

Figure 6.22: Average times in seconds of individual methods.

Figure 6.22 shows, that the only method which can keep up with the commonly used bilinear interpolation is the Zienkiwicz's interpolation, which is only few hundredths of seconds slower. The Natural Neighbour method is significantly slower, due to its quite complicated algorithm, which requires many arithmetic operations. Patch-based methods fall behind even more, as was expected. The Coons patch method is the slower one because of mathematically complicated computations required for defining the border curves of the patch (see part 3.4 for details).

7. Conclusion

Six different interpolation methods have been examined, described, implemented and tested. These methods represent three different possible approaches to interpolation of triangulations representing digital image.

First of these approaches is a classic interpolation of individual triangles, with the Bilinear interpolation being the known and most used representative of this approach. Zienkiewicz's interpolation was presented as an alternative for the Bilinear interpolation. It uses information in triangles surrounding the interpolated triangle to achieve better results and as experiments prove, it succeeds. Because its time consumption is only a little bit, in usual application insignificantly, bigger, it is a suggested replacement for the traditional Bilinear interpolation.

The Bezier patch and Coons patch methods presented another way of incorporating additional information then the information in a single triangle and that is the way of interpolating on larger, overlapping surfaces formed by more neighbouring triangles. They successfully remove the traces of the triangulation, making the image smoother then in case of Bilinear or Zienkiewicz's interpolation. However, this also implies unwanted blending of colour edges in the image. Unfortunately, another, initially not expected, flaws appeared. First of them is their tendency to produce artifacts. Therefore they are not suitable as universal interpolation methods. Nevertheless, this approach could be exploited more and used for partial interpolation of smooth areas or for interpolation of specially prepared triangulations, which would ensure that the artifacts won't appear. However, the second flaw of extremely large computational time is a big obstacle which would have to be eliminated somehow in order to make these methods useful.

The last approach used a dual configuration of the Delaunay triangulation, the Voronoi diagram. Although the piecewise linear interpolation proved to be a complete failure, the Natural Neighbour interpolation brought some promising results. Its time consumption is significantly higher then this of Bilinear or Zienkiewicz's interpolation, but it is still in bounds reasonable enough for using it. Though the quality of Natural Neighbour interpolation does not outperforms the Zienkiewicz's interpolation, it is close enough to take it into account in further development.

The implementation enables interpolation of both greyscale and coloured images. Coloured images may be represented by either separate triangulations for each colour component or the co-triangulation. The co-triangulations are more suitable when more effective compression is required, but their visual quality is usually worse then that of separate triangulation.

To conclude all the findings, it has to be admitted that although some improvements can be achieved when other then the Bilinear interpolation is used, they are not so major to make the triangulations the best tool for representation of digital image. If it is even possible to reach this goal, it is certain that the development of better interpolation methods has to be combined with the development of techniques for triangulation construction.

References

[1] Čermák, P.. *Výpočet vrstevnic na trojúhelníkové síti.* Diploma thesis, Faculty of Applied Sciences, University of West Bohemia in Pilsen, Czech Republic, 2002.

[2] Dyn, N. – Levin, D. – Rippa, S. *Data Dependent Triangulations for Piecewise Linear Interpolation*. IMA Journal of Numerical Analysis, Vol. 10 (1990), pp. 137 – 154.

[3] Kohout, J. *Delaunay Triangulation in Parallel and Distributed Environment*. Doctoral Thesis, Faculty of Applied Sciences, University of West Bohemia in Pilsen, Czech Republic, 2005.

[4] Kohout, J. *On Digital Image Representation by the Delaunay Triangulation.* PSIVT 2007, December 17 – 19 2007, pp. 826-840.

[5] Ledoux, H. – Gold C. *Interpolation as a tool for the modelling of threedimensional geoscientific datasets*. 4th ISPRS Workshop on Dynamic and Multi-dimensional GIS, 2005, pp. 79 – 84.

[6] Meyer, M. – Lee, H. – Barr, A. – Desbrun, M. *Generalized Barycentric Coordinates on Irregular Polygons*, Journal of Graphic Tools 7, 2002, pp. 13 – 22.

[7] Su, D. – Willis, P. *Image Interpolation by Pixel Level Data-Dependent Triangulation*. Computer Graphics Forum, Vol. 23 No. 2, June 2004, pp. 189 – 201.

[8] Sýkora, R. *Komprese barevných digitálních obrazů s využitím triangulace.* Bachelor thesis, Faculty of Applied Sciences, University of West Bohemia in Pilsen, Czech Republic, 2008.

[9] Wang, Z. – Bovik, A.C. – Sheikh, H.R. – Simoncelli, E.P. *Image Quality Assessment: From Error Visibility to Structural Similarity,* IEEE Transactions on Image Processing, vol. 13, no. 4, Apr. 2004, pp. 600 – 612.

[10] Weimer, H. – Warren, J. – Troutner, J. – Wiggins, W. – Shrout, J. *Efficient Co-Triangulation of Large Data Sets*. IEEE Visualization 98, pp. 119 – 126, 1998.

[11] http://en.wikipedia.org/wiki/Cohen-Sutherland

Appendix

- User manual
- Programmer’s manual
- Overview of appended material

User manual

This section will provide brief information on how to use the Interpolator program, which was created as a tool for testing of various interpolation methods. It requires Microsoft .NET framework to run.

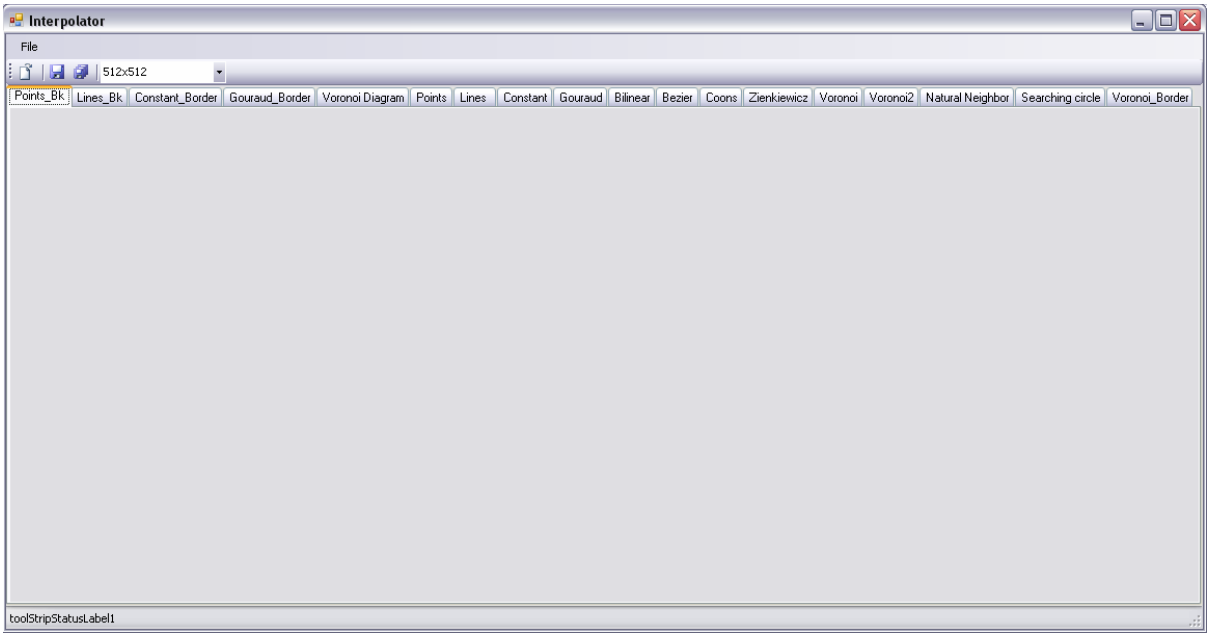

Figure I: The user interface of the Interpolator.

Figure I shows the user interface of the program. The main portion of the area is reserved for visualization of the loaded image. Each of the tab pages above means one interpolation method, or some additional visualization option, like visualization of the triangulation, visualization of the Voronoi diagram etc. When a tab page is selected, the respective image is rendered based on the loaded triangulation and presented.

To load a triangulation, use either the first icon on the tool strip menu (the "Open" icon) or the same option in the "File" menu. A dialog for opening a file will appear. Select the triangulation and open it.

The other two buttons on the tool strip menu are for saving either the image of the currently selected tab page, or images in all the tab pages. Note that as the second option needs to have all the images rendered, it can take some time. The last control on the tool strip menu is for changing the resolution of the loaded triangulation. Note that it is important to keep the same ration of width and height in order to retain Delaunay properties of the triangulation (in case that you are using Delaunay triangulation). If it is not kept, erroneous result might appear, especially in case of methods based on the Voronoi diagram.

One additional option is accessible through the "File" menu, the "Open and run many…". This option allows you to load more triangulations at the same time. Upon selecting this option, a dialog for opening multiple files will appear. Select all the triangulations you want to process and open them. They will be processed by the six methods described in this thesis, the resulting images and file containing the consumed times will be stored in

the root directory of the program in separate directories, one for each of the triangulation (having its name).

Programmer's manual

The program was originally created by Ing. Josef Kohout PhD., the supervisor of this thesis. He designed the application and implemented the basic visualization plug-ins, i.e. the visualization of the loaded triangulation, points of the triangulation, the Gouraud interpolation (bilinear interpolation using the scanline algorithm instead of barycentric coordinates) etc.

This implies he also implemented most of the data structures used by the application, like the Triangulation class, the PrjTriangulation class, Vertex and PrjVertex classes and the Triangle class. The author of this thesis implemented these data structures: VoronoiD class (for representation of the Voronoi diagram), the Vector class (for purposes of the Zienkiewicz's interpolation) and the Spline class (for the Coons patch method). All these structures are in module TriInterpolationCore.cs.

Module IncorporatedInterpolations.cs consists of classes respective to individual interpolation methods and other visualization techniques. Each tab page, mentioned in the User manual, is bound to one of these classes. They are inherited from the abstract TriInterpolationPlugin class (stored in its own module). These classes contain the information, whether the given method works with the triangulation or the Voronoi diagram and implement the Interpolate method. However, this method only does some initializations of the bitmap and then calls appropriate method from the RenderUnit class.

The RenderUnit is stored in module RenderUnit.cs. It contains the interpolation algorithms themselves. The interpolation methods take as an input the triangulation to be interpolated, its size and an array of the bitmap data to be filled.

Overview of appended material

The appended DVD contains several directories with additional material. Each of them contains a text file "readme.txt", which explains how the data in that directory are structured. These directories are:

- Additional software: programs used by the author for the experiments, but implemented by someone else. These programs include:
  - CompareImages – tool for image comparisons using the MSE, PSNR and SSIM metrics. Author: Ing. Josef Kohout PhD.
  - TriImgCompress – a tool for creating the triangulations from bitmap images. Author: Ing. Josef Kohout PhD.
  - TriImgCotriangulation – a tool for creating the co-triangulations from a given bitmap image. Author: Ing. Josef Kohout PhD.
  - TriImgAnalyzer – Multifunctional tool for extracting individual colour components from images and reconstruction of images from given components. Author: Bc. Radek Sýkora.
- Experiments: contains the data used for experiments and the results of these experiments.
- Interpolator: contains the actual program implemented (partly, see the Programmer's manual) by the author of this thesis and its source codes.

- Text: contains the text of this thesis as well as additional text directly related to its topic written by the author of this thesis. Namely its technical reports for subjects KIV/PRJ3 and KIV/PRJ5, which directly preceded the work done within the scope of this thesis and a paper presented in CESCG 2008 conference by the author.