© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 02 | 02 | April - June 2016

IMPLEMENTATION SCALING IMAGE USING BILINEAR INTERPOLATION USING MATLAB AND ANALYSING THE LIMITATIONS

[1]Ritika Wadhwa, Research Scholar, Department of CSE, PPIMT Hisar
[2]Mrs. Pallavi, Assistant professor, Department of CSE, PPIMT Hisar

Abstract: Two Dimensional computer graphics is computer-based generation of digital image/graphics—mostly from two-dimensional models such as two Dimensional geometric models, text, & digital image/graphics & by techniques specific to them. Word may stand for branch of computer science that comprises such techniques, or for models themselves. Two Dimensional computer graphics are mainly required in applications that were originally developed upon traditional printing & drawing technologies, such as technical drawing, advertising, typography, cartography, etc. In those applications, two-dimensional image/graphic is not just a representation of a real-world object, but an independent artifact with added semantic value, 2D models are therefore considered, because they give more direct control of image/graphic than three Dimensional computer graphics whose approach is considered more akin as compare to photography than to typography. In this paper we have make implementation of scaling using BILINEAR Interpolation and analyzed its limitations.

Keywords: Digital image, 2D, 3D, rotation, Scaling, Computer graphics, Matrix

I. INTRODUCTION

Two Dimensional computer graphics started in 1950s that is based on vector graphics devices. These were largely supplanted by raster-based devices in following decades. PostScript language & X Window System protocol were landmark developments in field. Two Dimensional graphics models may combine geometric models also known vector graphics, digital image/graphics also called raster graphics, text to be typeset is defined by content, font style & colour, position, size & orientation, mathematical functions & equation. Components can be modified & manipulated by two-dimensional geometric transformations such as rotation, translation, scaling. In object-oriented graphics, image/graphic is described indirectly by an object endowed with a self-rendering method a procedure which assigns colors to image/graphic pixels by an arbitrary algorithm. Complex models may be built by combining simpler objects, in paradigms of object-oriented programming.

2. Image Scaling

In computer graphics, **image scaling** is process of resizing a digital image. Scaling is a non-trivial process which involves a trade-off between efficiency, smoothness & sharpness. With bitmap graphics, as size of an image is reduced or enlarged, pixels which form image become increasingly visible, making image appear "soft" if pixels are averaged, or jagged if not. With vector graphics trade-off may be within processing power for re-rendering image, which may be obvious as slow re rendering with still graphics, or slower frame rate & frame skipping within computer animation.

Separately from fitting a smaller show area, image size is most commonly decreased (or subsampled or down sampled) within order to produce thumbnails. Enlarging an image (up sampling or interpolating) is generally common for creation lesser imagery fit a bigger screen. Within “zooming” a bitmap image, this is not possible to discover any more information within image than already exists, & image quality inevitably suffers. However, there are several methods of increasing number of pixels which an image contains, which evens out appearance of original pixels.

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 02 | 02 | April - June 2016

Algorithm

Two standard scaling algorithms are bilinear & bicubic interpolation. Filters like these work by interpolating pixel color values, introducing a continuous transition into output even where original material has discrete transitions. Although this is required for continuous-tone images, many algorithms reduce contrast (sharp edges) within a way which may be undesirable for line art.

Nearest-neighbor interpolation preserves these sharp edges, but this increases aliasing (or jaggies; where diagonal lines & curves appear pixelated). Several approaches have been developed which attempt to improve for bitmap art by interpolating areas of incessant tone, preserve sharpness of horizontal & vertical lines & smooth all other curves.

3. BILINEAR FILTERING

It is a texture filtering method used to smooth textures when showed larger or smaller than they really are. Maximum of time, when drawing a textured shape on screen, texture is not showed exactly as this is stored, without any distortion. since of this, most pixels will end up needing to use a point on texture which is "between" texels, assuming texels are points (as opposed to, say, squares) within middle (or on upper left corner, or anywhere else; this does not matter, as long as this is consistent) of their respective "cells". In Bilinear filtering uses these plugs to perform bilinear interpolation between four texels nearest to point which pixel represents (in middle or upper left of pixel, usually).

Formula

In a mathematical context, bilinear interpolation is problem of finding a function f(x,y) of form

$$f(x,y) = c_{11}xy + c_{10}x + c_{01}y + c_{00}$$

satisfying

$$f(x_1,y_1) = z_{11}$$
$$f(x_1,y_2) = z_{12}$$
$$f(x_2,y_1) = z_{21}$$
$$f(x_2,y_2) = z_{22}$$

The usual, & generally computationally least expensive way to compute f is through linear interpolation used 2 times, for example to compute two functions f_1 and f_2 satisfying

$$f_1(y_1) = z_{11}$$
$$f_1(y_2) = z_{12}$$
$$f_2(y_1) = z_{21}$$
$$f_2(y_2) = z_{22}$$

and then to combine these functions (which are linear within y) into one function f satisfying

$$f(x_1,y) = f_1(y)$$
$$f(x_2,y) = f_2(y)$$

In computer graphics, bilinear filtering is generally performed on a texture during texture mapping, or on a bitmap during resizing. within both cases, source data (texture or bitmap) may be seen as a two-dimensional array of values z_{ij}, or several (usually three) of these within case of full-color data. data points used within bilinear filtering are 2x2 points surrounding location for which color is to be interpolated.

Furthermore, one does not have to compute actual coefficients of function f; computing *value* $f(x,y)$ is enough.

Largest integer not larger than x shall be called $[x]$, & fractional part of x shall be $\{x\}$. Then, $x = [x] + \{x\}$, & $\{x\} < 1$. We have $x_1 = [x]$, $x_2 = [x] + 1$, $y_1 = [y]$, $y_2 = [y] + 1$. data points used for interpolation are taken from texture / bitmap & assigned to z_{11}, z_{12}, z_{21}, & z_{22}.

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 02 | 02 | April - June 2016

$f_1(y_1) = z_{11}$, $f_1(y_2) = z_{12}$ are two data points for f_1 subtracting former from latter yields

$$f_1(y_2) - f_1(y_1) = z_{12} - z_{11}$$

Because f_1 is linear, its derivative is constant & equal to

$$(z_{12} - z_{11})/(y_2 - y_1) = z_{12} - z_{11}$$

Because $f_1(y_1) = z_{11}$,

$$f_1(y_1 + \{y\}) = z_{11} + \{y\}(z_{12} - z_{11})$$

and similarly,

$$f_2(y_1 + \{y\}) = z_{21} + \{y\}(z_{22} - z_{21})$$

Because $y_1 + \{y\} = y$, we have computed endpoints $f_1(y)$ and $f_2(y)$ needed for second interpolation step.

The second step is to compute $f(x, y)$, which may be accomplished by very formula we used for computing intermediate values:

$$f(x, y) = f_1(y) + \{x\}(f_2(y) - f_1(y))$$

In case of scaling, y remains constant within same line of rescaled image, & storing intermediate results & reusing them for calculation of next pixel may lead to significant savings. Similar savings may be achieved with all "bi" kinds of filtering, i.e. those which may be conveyed as two passes of one-dimensional filtering.

In case of texture mapping, a constant x or y is rarely if ever encountered, & since today's (2000+) graphics hardware is highly parallelized, there would be no time savings anyway.

Another way of writing bilinear interpolation formula is

$$f(x, y) = (1 - \{x\})((1 - \{y\})z_{11} + \{y\}z_{12}) + \{x\}((1 - \{y$$

4. IMPLEMENTATION OF SCALING IMAGE USING BILINEAR INTERPOLATION

```
function [out] = bilinearInterpolation(im, out_dims)
    %// Get some necessary variables first
    in_rows = size(im,1);
    in_cols = size(im,2);
    out_rows = out_dims(1);
    out_cols = out_dims(2);
     %// Let S_R = R / R'
    S_R = in_rows / out_rows;
    %// Let S_C = C / C'
    S_C = in_cols / out_cols;

    %// Define grid of co-ordinates within our image
    %// Generate (x,y) pairs for each point within our image
    [cf, rf] = meshgrid(1 : out_cols, 1 : out_rows);
     %// Let r_f = r'*S_R for r = 1,...,R'
    %// Let c_f = c'*S_C for c = 1,...,C'
    rf = rf * S_R;
    cf = cf * S_C;
     %// Let r = floor(rf) & c = floor(cf)
    r = floor(rf);
    c = floor(cf);
     %// Any values out of range, cap
    r(r < 1) = 1;
    c(c < 1) = 1;
    r(r > in_rows - 1) = in_rows - 1;
    c(c > in_cols - 1) = in_cols - 1;

    %// Let delta_R = rf - r & delta_C = cf - c
    delta_R = rf - r;
    delta_C = cf - c;
     %// Final line of algorithm
    %// Get column major indices for each point we wish
    %// to access
    in1_ind = sub2ind([in_rows, in_cols], r, c);
    in2_ind = sub2ind([in_rows, in_cols], r+1,c);
    in3_ind = sub2ind([in_rows, in_cols], r, c+1);
    in4_ind = sub2ind([in_rows, in_cols], r+1, c+1);
```

```
%// Now interpolate
%// Go through each channel for case of colour
%// Create output image that is same class as input
out = zeros(out_rows, out_cols, size(im, 3));
out = cast(out, class(im));

for idx = 1 : size(im, 3)
    chan = double(im(:,:,idx)); %// Get i'th channel
    %// Interpolate channel
    tmp = chan(in1_ind).*(1 - delta_R).*(1 -
delta_C) + ...
            chan(in2_ind).*(delta_R).*(1 -
delta_C) + ...
            chan(in3_ind).*(1 -
delta_R).*(delta_C) + ...
            chan(in4_ind).*(delta_R).*(delta_C);
    out(:,:,idx) = cast(tmp, class(im));
end
```

Calling Function to find Bilinear Interpolation

```
im=imread('ds.jp')
out = bilinearInterpolation(im, [270 396]);
figure;
imshow(im);
figure;
imshow(out);
```

Following output is displayed

Original image

Resized image

5. SCOPE AND CONCLUSION

Bilinear filtering is rather accurate until scaling of texture gets below half or above double original size of texture - which is, if any texture was 256 pixels in separately direction, scaling this to below 128 or above 512 pixels may make texture look bad, since of missing pixels or too much smoothness. Often, mipmapping is used to provide a scaled-down version of texture for better performance; however, transition between two differently-sized mipmaps on a texture in perspective using bilinear filtering may be very abrupt. Trilinear filtering, though somewhat more complex, may make this transition smooth through.

For any quick demo of how a texel may be missing from a filtered texture, here's a list of numbers representing centers of boxes from an 8-texel-wide texture (in black & red), intermingled with numbers from cores of boxes from a three texel wide down sampled texture (in blue). red numbers represent texels which would not be used in calculating 3-texel texture at all.

0.0625, 0.1667, 0.1875, 0.3125, 0.4375, 0.5000, 0.5625, 0.6875, 0.8125, 0.8333, 0.9375

Special cases

Textures aren't infinite, within general, & sometimes one ends up with a pixel coordinate which lies outside grid of texel coordinates. There are a few ways to handle this:

- Wrap texture, so which last texel within a row also comes right before first, & last texel in a column also comes right above first. This works best when texture is being tiled.
- Make area outside texture all one color. This may be of use for a texture designed to be laid over a solid background or to be transparent.
- Repeat edge texels out to infinity. This works best if texture is not designed to be repeated.

6. REFERENCES

1. Dudgeon, D.E. & R.M. Mersereau, Multidimensional Digital Signal Processing. 1984, Englewood Cliffs, New Jersey: Prentice-Hall.

2. Castleman, K.R., Digital Image/graphic Processing. Second ed. 1996, Englewood Cliffs, New Jersey:

3. Oppenheim, A.V., A.S. Willsky, & I.T. Young, Systems & Signals. 1983, Englewood

4. Papoulis, A., Systems & Transforms with Applications in Optics. 1968, New York:

5. Russ, J.C., Image/graphic Processing Handbook. Second ed. 1995, Boca Raton, Florida: CRC

6. Giardina, C.R. & E.R. Dougherty, Morphological Methods in Image/graphic & Signal Processing. 1988, Englewood Cliffs, New Jersey: Prentice-Hall. 321.

7. Gonzalez, R.C. & R.E. Woods, Digital Image/graphic Processing. 1992, Reading, Massachusetts:

8. Goodman, J.W., Introduction to Fourier Optics. McGraw-Hill Physical & Quantum Electronics Series. 1968, New York: McGraw-Hill. 287.

9. Heijmans, H.J.A.M., Morphological Image/graphic Operators. Advances in Electronics & Electron Physics. 1994, Boston: Academic Press.

10. Hunt, R.W.G., Reproduction of Colour in Photography, Printing & Television,. Fourth ed. 1987, Tolworth, England: Fountain Press.

11. Freeman, H., Boundary encoding & processing, in Picture Processing & Psychopictorics, B.S. Lipkin & A. Rosenfeld, Editors. 1970, Academic Press: New York. p. 241-266.

12. Stockham, T.G., Image/graphic Processing in Context of a Visual Model. Proc. IEEE, 1972. 60:

13. Murch, G.M., Visual & Auditory Perception. 1973, New York: Bobbs-Merrill Company,

14. Frisby, J.P., Seeing: Illusion, Brain & Mind. 1980, Oxford, England: Oxford University

15. Blakemore, C. & F.W.C. Campbell, On existence of neurons in human visual system selectively sensitive to orientation & size of retinal image/graphics. J. Physiology, 1969.

16. Born, M. & E. Wolf, Principles of Optics. Sixth ed. 1980, Oxford: Pergamon Press.

17. Young, I.T., Quantitative Microscopy. IEEE Engineering in Medicine & Biology, 1996.

18. Dorst, L. & A.W.M. Smeulders, Length estimators compared, in Pattern Recognition in Practice II, E.S. Gelsema & L.N. Kanal, Editors. 1986, Elsevier Science: Amsterdam. p. 73-80.

19. Young, I.T., Sampling density & quantitative microscopy. Analytical & Quantitative Cytology & Histology, 1988. 10(4): p. 269-275.

20. Kulpa, Z., Area & perimeter measurement of blobs in discrete binary pictures. Computer Vision, Graphics & Image/graphic Processing, 1977. 6: p. 434-454.

21. Vossepoel, A.M. & A.W.M. Smeulders, Vector code probabilities & metrication error in representation of straight lines of finite length. Computer Graphics & Image/graphic Processing,

22. Photometrics Ltd., Signal Processing & Noise, in Series 200 CCD Cameras Manual. 1990: