Optimal Depth Buffer for Low-Cost Graphics Hardware

Eugene Lapidous*, Guofang Jiao*

Trident Microsystems Inc.

Abstract

3D applications using hardware depth buffers for visibility testing are confronted with multiple choices of buffer types, sizes and formats. Some of the options are not exposed through 3D API or may be used by the driver without application's knowledge. As a result, it becomes increasingly difficult to select depth buffer optimal for desired balance between performance and precision.

In this paper we provide comparative evaluation of depth precision for main depth buffer types with different size and format combinations. Results indicate that integer storage is preferred for some buffer types, while others achieve maximal depth resolution with floating-point format optimized for known scene parameters. We propose to give 3D applications full control of the depth buffer optimization by supporting multiple storage formats with the same buffer size and exposing them in 3D API.

In the search for a unified depth buffer solution, we describe new type of the depth buffer and compare it with other options. Complementary floating-point Z buffer is a combination of a reversed-direction Z buffer and an optimal floating-point storage format. Non-linear mapping and storage format compensate each other's effect on the depth precision; as a result, depth errors become significantly less dependent on the eye-space distance, improving depth resolution by the orders of magnitude in comparison with standard Z buffer. Results show that complementary Z buffer is also superior to inverse W buffer at any storage size. At 16 and 24 bits/pixel, average depth errors of complementary Z buffer remain 2 times larger than for true W buffer utilizing expensive high-precision per-pixel division. However, it provides absolutely best precision at 32 bits/pixel, when errors are limited by floating-point per-vertex input.

Results suggest that complementary floating-point Z buffer can be considered as a candidate for replacement of both screen Z and inverse W buffers, at the same time making hardware investment in the true W buffer support less attractive.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture – raster display devices; I.3.3 [Computer Graphics] Picture/Image Generation – display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – visible surface algorithms.

Additional Key Words and Phrases: depth buffer, Z buffer, screen Z, W buffer, depth precision, visibility error.

* 2450 Walsh Avenue, Santa Clara, CA 95051
{eugene,jgf}@tridmicr.com

1 INTRODUCTION

Depth buffer algorithms are used for visibility testing by the vast majority of modern 3D graphics accelerators. Depth buffer architecture is shaped by the conflicting requirements of precision, buffer size, memory bandwidth and complexity of per-pixel conversion to and from storage format. High-end graphics accelerators usually support depth buffers with 32 or 24 bits/pixel[1,2,3]; low-cost 3D hardware may store depth values at 16, 20 or 24 bits/pixel[4]. Other depth formats such as 26-bit floating-point value with 6-bit exponent, are also proposed[5].

To balance precision and performance, the same 3D accelerator may support multiple combinations of per-pixel depth and buffer type, using hardware-specific storage formats.

As a result of available hardware options, driver and an application (to the extent of exposure through 3D API) are confronted with multiple choices that affect performance and quality of the depth buffer.

Best-defined options are per-pixel buffer size (usually 16, 24 or 32 bits) and standard Z buffer type, storing result of the projective normalized device coordinates. Constraints of such transformation are usually normalization to [0,1] range; preservation of lines and planes; preservation of the sign of distance change[6]. Non-linear mapping under these conditions results in a loss of roughly $\log_2(r)$ bits of the depth precision for a given ratio r of the distances to the far and near clipping planes[7]. This precision decrease causes severe visual artifacts in the open environments with large dynamic range r, typical for the flight simulators and games[8].

Better depth precision at the same buffer size is achieved if normalized eye-object distance is stored instead of the screen-space depth[9]. Projective transformation to homogeneous coordinates usually produces value W proportional to the eye-object distance; inverse of this value is interpolated for perspective-correct texture mapping. If W value, normalized to the distance to the far plane, is computed with depth-buffer precision and stored in the integer format, it guarantees linear mapping of the eye-object distance to the depth buffer (W buffer). However, this operation requires costly high-precision per-pixel division, with more bits that are needed for perspective correction. Also, resolution of the W buffer drops for scenes with ratios of the distances to the far and near planes close to 1, because most of the depth range becomes unused.

To avoid additional hardware cost or performance penalty, depth buffer may store interpolated value of rhW (1/W) computed with high precision before per-pixel perspective division. Inverse W buffer, also known as 1/Z buffer, is already supported by at least one family of 3D graphics accelerators[10]. To authors knowledge, no 3D API currently allows to make a distinction between true and inverse W buffer.

Least exposed feature of the depth buffer is the storage format. Most popular options are integer and variations of the floating-point format: 16 bits/pixel with 12 bits of mantissa and 4 bits of exponent; 24 bits/pixel storing IEEE float without 8 last bits of mantissa; 32 bits/pixel storing full-precision IEEE float.

Selection of the depth buffer type can be done by an application (to the extent supported by 3D API) or by the driver[11]; selection of the storage format is usually defined by the choice of graphics hardware.

Imprecise characteristics of these multiple options provide no guarantee that selected depth buffer has good precision/performance balance for the current application. To find actual depth error distribution in the view volume, application has to rely on the run-time tests or previously collected data for different driver-hardware combinations.

In this paper we propose to simplify available options by using optimal depth buffer with known set of formats. To qualify, such buffer has to work with current 3D APIs and majority of graphics hardware, providing superior precision at the low implementation cost.

Next section contains comparative evaluation of depth precision for popular depth buffers with multiple size and format options. After that, we describe new depth buffer type and compare it with others as a candidate for an optimal solution.

2 DEPTH PRECISION EVALUATION

2.1 Methodology

Main factors affecting depth resolution in the eye space are mapping of the eye-object distance (Ze) to the depth in the normalized device coordinates (D) and precision of the format used to represent D:

$$\delta Ze(Ze) = \frac{dZe}{dD} * \delta D(D) \qquad (1)$$

$D =$ If internal hardware operations are performed with sufficient precision, error of D representation can be estimated as largest of 2 factors:

1. Per-vertex error of D representation after computation of normalized device coordinates. Most popular input format for vertex coordinates is IEEE float; increasing its size may cause performance and bandwidth problems.

2. Per-pixel error of D representation in the depth buffer, dependent on its size and format.

For per-pixel D representation, we evaluate floating-point format with n bits of exponent (e) and m bits of mantissa (f):

$$\begin{cases} 2^{e-2^n} * 1.f \cdots (2^n > e > 0) \\ 2^{1-2^n} * 0.f \cdots (e = 0) \end{cases} \qquad (2)$$

In further analysis we assume that all values stored in the depth buffer have the same sign. Sign bit, if present, doesn't affect precision but decreases total number of bits for mantissa and exponent at the same per-pixel buffer size. Handling of corner cases can depend on implementation without significant effect on estimated precision. Description also covers integer format (n = 0, e = 0), which by definition (2) is equivalent to floating-point format with 1 bit of exponent (n = 1).

Normalized mapping functions D(Ze) are different for 3 types of depth buffers and depend on the values of the distance to the far (f) and near (d) clipping planes in the eye space:

Z buffer: $$D = \frac{f}{f-d} * (1 - \frac{d}{Ze}) \qquad (3)$$

W buffer: $$D = \frac{Ze}{f} \qquad (4)$$

Inverse W buffer: $$D = \frac{d}{Ze} \qquad (5)$$

Average depth precision in the eye space depends on the buffer type, representation error and sampling distribution. Because representation error is discrete, we computed average normalized depth error for a finite number of samples:

$$\overline{\delta Ze} = \frac{\sum_{Ze=d}^{Ze=f} \delta Ze(Ze) * S(Ze)}{(f-d) * \sum_{Ze=d}^{Ze=f} S(Ze)} \qquad (6)$$

where S(Ze) is a number of samples taken at the distance Ze, and dZe(Ze) is a current depth error dependent on mapping function (3..5) and maximum of per-vertex (dDv) and per-pixel (dDp) representation errors:

$$\delta Ze(Ze) = \left| Ze(D + \max[dDv, dDp]) - Ze(D) \right| \qquad (7)$$

Results are obtained for uniform (S(d) = S(f)) and linearly increasing (S(d) = 0) sampling distributions, 16, 24, and 32 bits/pixel.

2.2 Results

Before comparing different types of depth buffers, we evaluate optimal storage format for each type and storage size.

For realistic sampling distributions, integer storage is always the best choice for Z- and W buffer; Fig.1, 2 show examples for 16-bit buffer with uniform sampling. Legend indicates number of mantissa and exponent bits for each data set (16-bit mantissa corresponds to integer format). The same trend is valid for 24 bits/pixel storage.

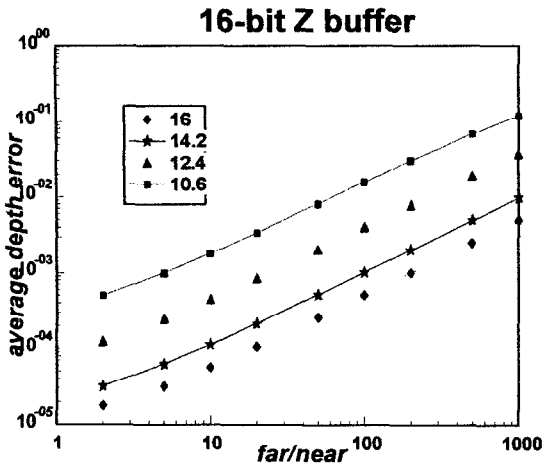


Figure 1

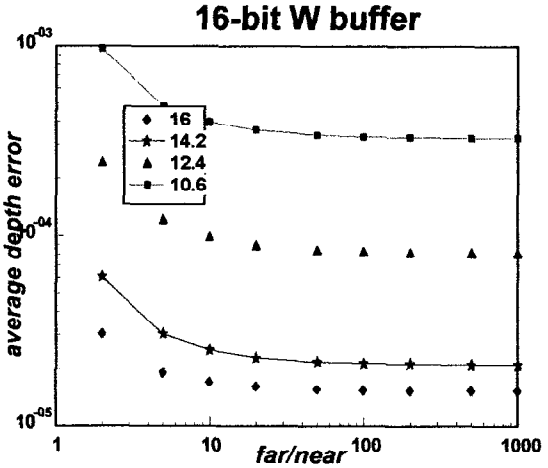


Figure 2

Floating-point storage is optimal for inverse W buffer; Fig. 3,4 show results for 16 and 24 bits/pixel storage sizes. Absolute floating-point error for small D compensates increased non-linear mapping at large eye-object distance, improving precision in comparison with Z buffer. Increase of the sampling frequency in the direction of the far clipping plane increases average error by 20..45% (Fig.5), but relation between different data formats remains unchanged.

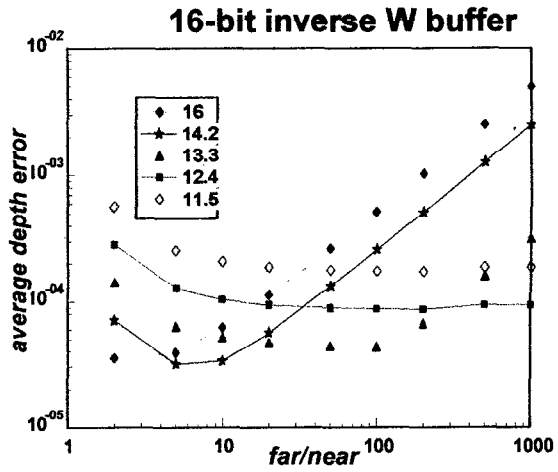


Figure 3

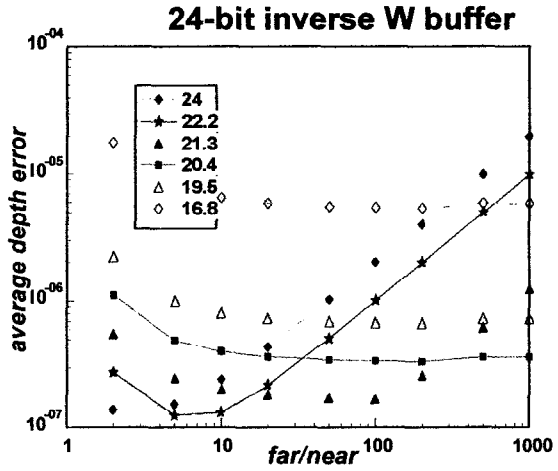


Figure 4

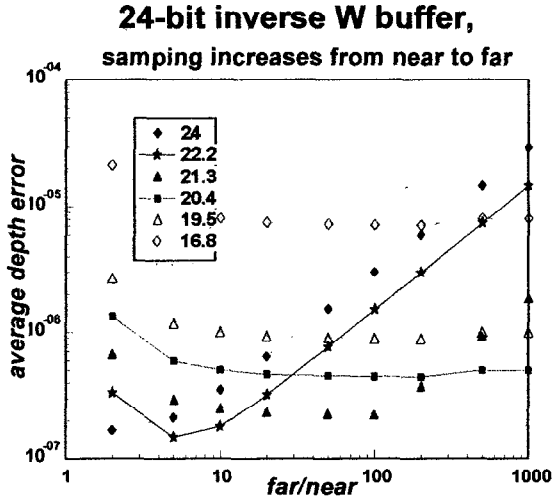


Figure 5

Results show that floating-point format with 8 bits of exponent, routinely used with 24-bit depth buffers, is far from optimal for inverse W buffer; best precision is achieved with only 2..4 bits of exponent, dependent on the sampling distribution and f/d ratio.

If only one storage format can be used for the buffer of each size, 3 or 4 bits of exponent are the best for inverse W buffer at 16 and 24 bits/pixel. To achieve maximal precision improvement, we propose to provide hardware support for a set of floating-point storage formats with different sizes of exponent. Driver or an application will select best format based on the dynamic range of the scene and area of interest. As shown on Fig.3-5, 4 bits of exponent can be selected for scenes with f/d > 200; 3 bits of exponent are optimal for 200 > f/d > 20. Sharp anomalies in sampling distribution may change optimal selection: for instance, if area of interest is very close to the near plane, integer format may be the best even for large ratios f/d.

Flexible selection of storage format increases average depth precision of inverse W buffer and decreases it's dependency on the f/d ratio. However, our results show that minimal average errors for 16-bit and 24-bit inverse W buffer remain approximately 6 times larger than for the true W buffer. As in case of the true W buffer, inverse W buffer can't be used if W isn't available or isn't proportional to the eye-space distance. Therefore, inverse W buffer may not be precise enough to be used instead of the true W buffer, and may not always be used to replace standard Z buffer.

3 COMPLEMENTARY Z BUFFER

Optimal depth buffer has to support linear mapping from eye-space distance to the storage, providing precision similar to the true W buffer without associated hardware complexity and software limitations.

Best precision achieved by the true W buffer is a result of the linear mapping from the eye-space distance to the integer storage format. Both factors in the equation (1) are kept constant: mapping from the eye-space to the screen is linear due to the format definition (4); integer format guarantees that precision is independent from the stored value.

We propose to maintain quasi-linear mapping by using a combination of simple mapping and format precision functions that compensate each other's non-linearity, instead of requiring both of them to be linear. To achieve this goal, new depth buffer type combines linear transformation of standard Z values and floating-point storage.

Screen depth D is computed using all Z buffer mapping constraints except distance change sign; result is complementary to the screen Z in the same range [0,1]:

$$D = 1 - Zs = \frac{d}{f-d} * (\frac{f}{Ze} - 1) \qquad (8)$$

To account for the different sign of distance change, we reverse depth test function, sign of Z bias and polygon offset. Depth value is stored in the floating-point format [2].

As in the case of standard Z buffer (3), non-linear mapping error is largest when Ze close to f. At the same time, floating-point format precision error is smallest when D is close to 0. Because function (8) maps far clipping plane (Ze = f) to D=0, these 2 factors compensate each other in the areas distant from the camera, which contribute most to the average error.

Such compensation is effective enough to make depth error almost independent of the distance to the camera, making it orders of magnitude smaller than for a standard Z buffer of the same size (Fig.6).

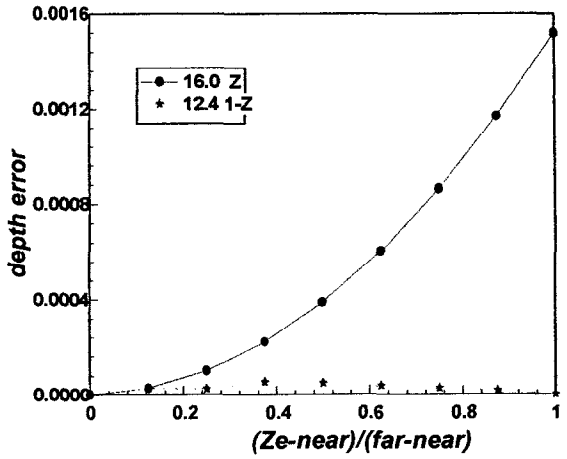


Figure 6

As in case of inverse W buffer, there is a set of optimal floating-point formats minimizing average error of the complementary Z buffer for different ranges of f/d ratio; results for 16 bits/pixel and uniform sampling are presented on Fig.7.

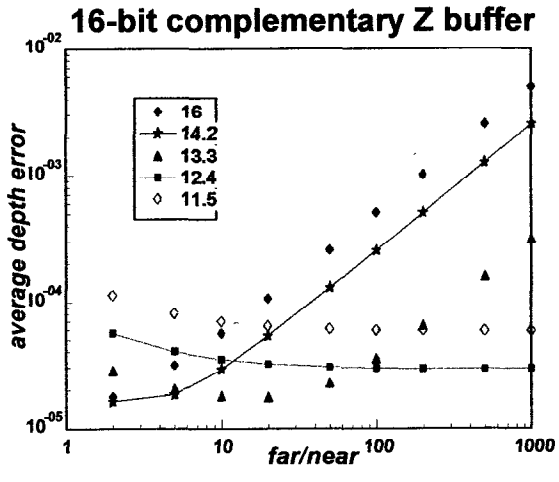
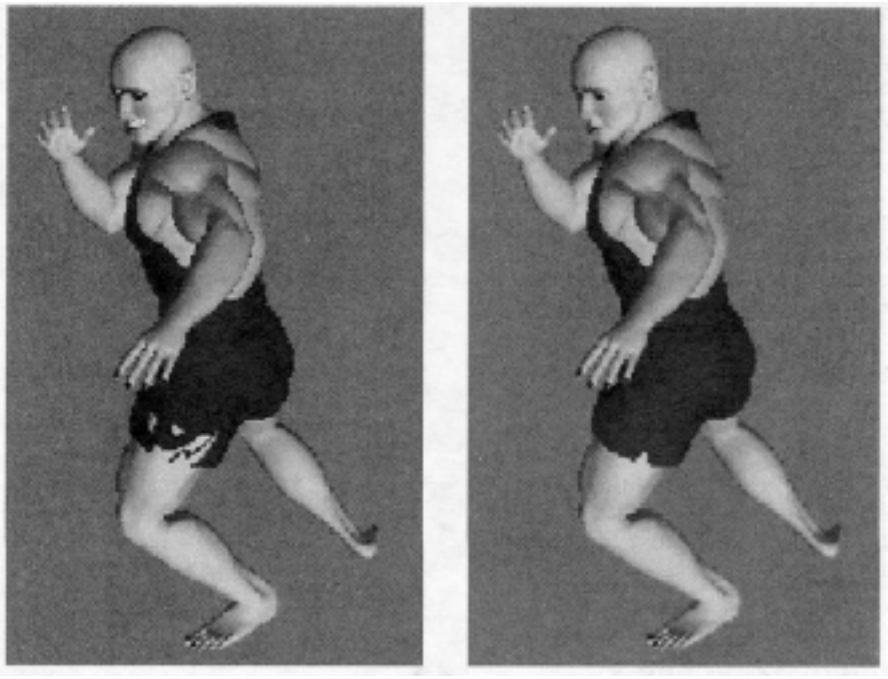


Figure 7



a b

Figure 8

As an example of precision improvement, Fig.8 shows the same character in the scene with ratio f/d=1000, rendered using 2 different types of depth buffers. 16-bit Z buffer causes visible body-cloth intersections and depth-sorting errors of the elements of character's head (Fig.8a); 16-bit complementary floating-point Z buffer resolves the same scene without any depth-related visual artifacts (Fig.8b).

Fig.9 shows comparison of minimal depth errors of all types of depth buffers described above, computed for optimal data formats at 16 bits/pixel, constant and increasing sampling distributions. Data for 24 bits/pixel are presented on Fig.10. Results for W buffer are shown once because they don't depend on sampling mode.

Presented data show that complementary floating point Z buffer is up to 3..4 times more precise than inverse W but still has 2 times larger average error than true W buffer. Results for complementary Z buffer are also significantly less dependent on the sampling distribution than in the cases of Z and inverse W buffers. In general, 16- and 24-bit complementary Z buffer is

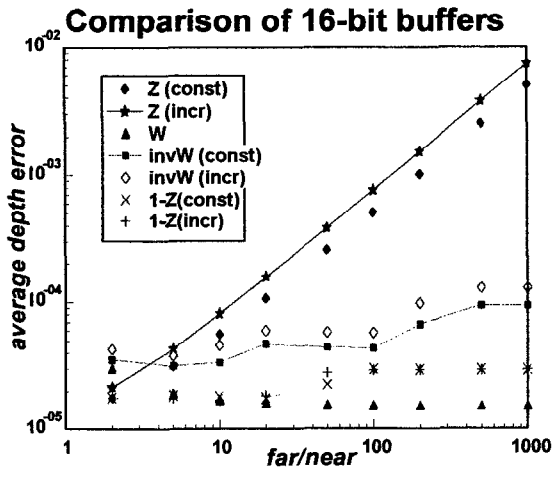


Figure 9

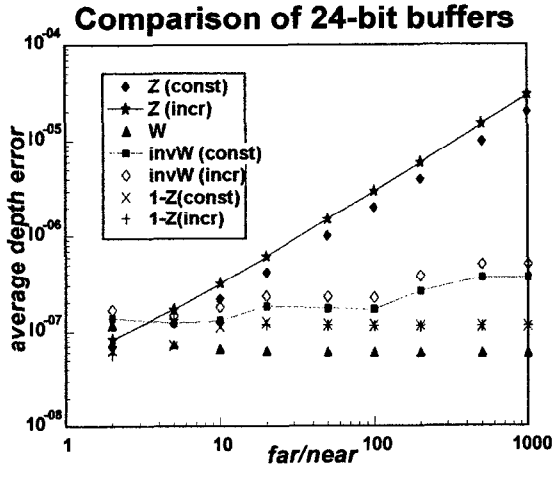


Figure 10

always as good or better than both Z and inverse W buffers, closing the distance with true W buffer.

Main reason for complementary Z advantage over inverse W is better utilization of the area close to D = 0. Inverse W buffer (5) maps Ze = f to D = f/d, preventing floating-point storage from compensating non-linear distribution as effectively as it does for complementary Z at D = 0. Inverse W isn't mapped to the area close to D = 0 to avoid affecting precision of the divide during perspective-correct texturing. Per-pixel offset –d/f may increase depth precision of inverse W with additional hardware support, but it will still remain lower than for complementary Z at small f/d ratios.

Relationship between different buffer types changes with further increase of the storage size; Fig.11 shows results for 32 bits/pixel. First, results for 32-bit Z buffer at f/d > 2 are within 1% margin of difference from the results for 24-bit Z buffer. Second, results for the true and inverse W buffers become almost identical even if one is in 32-bit integer format, while other has 8-bit exponent. Third, complementary Z buffer demonstrates the best precision – 3 times better than W buffer.

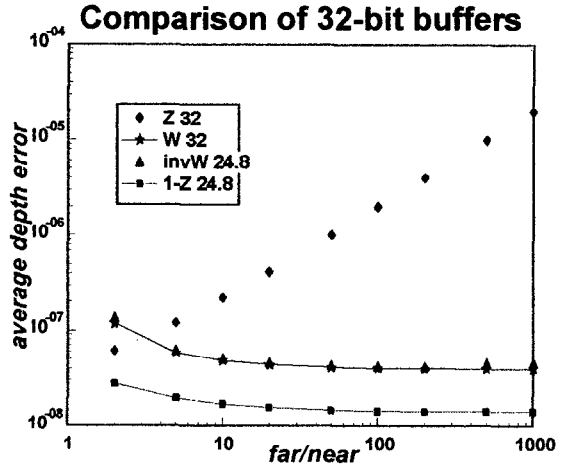


Figure 11

These changes are caused by the IEEE floating-point precision of the input per-vertex data that starts to provide main contribution to the average depth error with increase of the per-pixel storage size. As our results show, Z buffer precision depends mainly on the size of mantissa and is the best for integer formats. Therefore, 32-bit format with 23 bits of mantissa isn't better for Z than 24-bit integer. Floating-point input also decreases precision of W buffer, making it non-linear. Complementary Z buffer takes best advantage of the floating-point format because it uses it to compensate non-linearity of projection mapping. IEEE floating-point isn't optimal for complementary Z; our results show that the same precision can be achieved with 24.4 28-bit format. However, this increase is enough to make complementary Z the best buffer for high depth resolutions.

4 DISCUSSION AND CONCLUSIONS

As shown by our results, both inverse W and complementary Z buffers will benefit from support for multiple storage formats with 0..4 bits of exponent. Floating-point storage with 4 bits of exponent and 12 bits of mantissa is already supported by popular family of 3D accelerators[6]. While designed for inverse W buffer, this and other hardware with similar features can provide improved depth precision by using the same floating-point storage for Z values (after conversion to 1-Z). Set of different storage formats will be implemented in the future hardware designs.

When multiple storage formats for the same buffer size become available, driver will be able to select best format based on known f/d ratio and guessed sampling distribution. However, it may also be useful to expose multiple depth storage formats in the new versions of 3D APIs. It would allow application to make a choice of the depth buffer format based on it's better knowledge of the area of interest (however, it is less important for complementary Z buffer because it's less dependent on the sampling distribution). Additional benefit of such exposure would be a well-defined set of storage formats for compliance testing and mixed software/hardware access.

In comparison with inverse W, complementary floating-point Z buffer is significantly easier to use: it does not require W values to be proportional to the eye-space distance; doesn't need f and d values for normalization; doesn't loose precision for very small f/d ratios. It also can be utilized without additional support from 3D API, just by changing projection matrix. For instance, it can be done by using OpenGL function glDepthRange to set zNear = 1, zFar = 0. If projection matrix isn't available, operation Z = 1-Z can be performed by the driver or in the graphics hardware.

Because complementary Z buffer is always as good or better than Z or inverse W, it can be always enabled, even if storage format isn't optimal or unknown.

Below is the summary of main advantages of the complementary floating-point Z buffer:

1. The same or better precision than for Z and inverse W buffers at all storage sizes; better precision than for all types of depth buffers, including true W buffer, at 32 bit/pixel storage.
2. Simple implementations do not require additional support by 3D API and can use existing 3D hardware.
3. Optimal storage format is less dependent on the sampling distribution and therefore easier to select than in the case of inverse W buffer.

Our main conclusion is that complementary floating-point Z buffer may be a good candidate for replacement of both screen Z and inverse W buffers, at the same time making hardware investment in true W buffer support less attractive. It is also more suitable for next-generation hardware, providing better precision than W buffer when per-pixel depth storage exceeds 24 bits.

We consider major contributions of the paper to be following:

- Comparative precision analysis for popular types, sizes and formats of the depth buffers.
- Development of the new type of the depth buffer – complementary floating-point Z buffer - proposed as a candidate for a unified depth buffer solution
- Demonstration of depth precision improvement when multiple floating-point formats are supported for the depth buffer of the same size and optimal format is selected based on known scene parameters.
- Analysis of the maximal depth buffer precision limited by per-vertex data format.

5 Acknowledgements

We'd like to thank Yihui Wu and Tim Wilson from Trident Microsystems Architecture team for their help in developing software implementations of proposed algorithms.

References

[1] K.Akeley . Reality Engine Graphics. *Proceedings of SIGGRAPH '93* (Anaheim, California, August 1-6, 1993), In *Computer Graphics* Proceedings, Annual Conference Series, 1993, pages 109-116. ACM SIGGRAPH,, New York 1993.

[2] M.F.Deering, S.R.Nelson. Leo: A System for Cost Effective 3D Shaded Graphics. *Proceedings of SIGGRAPH '93* (Anaheim, California, August 1-6, 1993), In *Computer Graphics* Proceedings, Annual Conference Series, 1993, pages 101-108. ACM SIGGRAPH,, New York 1993.

[3] M.F.Deering, S.A.Schlapp, M.G.Lavelle, FBRAM: A new Form of Memory Optimized for 3D Graphics. *Proceedings of SIGGRAPH '94* (Orlando, Florida, July 24-29, 1994), In *Computer Graphics* Proceedings, Annual Conference Series, 1994, pages 167-174. ACM SIGGRAPH, July 1994.

[4] J.Torborg, J.T.Kajiva. Talisman: Commodity Realtime 3D Graphics for the PC. *Proceedings of SIGGRAPH '96*(New Orleans, LA, August 4-9, 1996), In *Computer Graphics* Proceedings, Annual Conference Series, 1996, pages 353-364. ACM SIGGRAPH,,1996.

[5] B.Rivard, S.Winner, M.Kelley, B.Pease, A.Yen Hardware Accelerated Rendering of Antialiasing Using a Modified A-Buffer Algorithm. *Proceedings of SIGGRAPH '97* (Los Angeles, California, August 3-8, 1997), In *Computer Graphics* Proceedings, Annual Conference Series, 1997, pages 307-316. ACM SIGGRAPH,, 1997.

[6] W.M. Newmann, R.F. Sproull Principles of Interactive Computer Graphics. 1981 New York:McGraw-Hill

[7] R. Kempf, C.Frazier, editors. OpenGL Reference Manual, Second Edition. The Official Reference Document to OpenGL, Version 1..1. 1997. Addison-Wesley, p.164

[8] J.Swarovsky. Extreme Detail Graphics. *Proceedings of Game Developers Conference'1999*, (San Jose, California, March 16-18, 1999) , pages 899-904

[9] Microsoft Corp. What are Depth Buffers? DirectX 6 Software Development Kit, 1998. http://msdn.microsoft.com/library/sdkdoc/directx/imover_4xwk.htm

[10] G.Tarolly. Gary Tarolli on 32bit vs. 16bit color. 3df/x Interactive Inc., 1999, http://www.3dfx.com/view_io.asp?ID=144

[11] R.Huddy High-performance D3D, Advanced D3D Programming. Presentation at the *Game Developers Conference'1999*, (San Jose, California, March 16-18, 1999) NVIDIA Corporation, March 1999, http://www.nvidia.com/Marketing/Company/Pages.nsf/pages/DevHome