**19**

**eXtended Perspective Shadow Maps**

*Jon Olick*

In a dynamic scene, it is often desirable to render shadows dynamically. Many approaches have been used over time to render dynamic shadows. Stencil shadows and variants have approached the problem from a geometrical perspective. Stencil shadows, however, generally have issues with excessive fill rate and triangle throughput. Many solutions to these problems have been proposed and used in production games, but most still have problems with alpha-tested triangles and also require closed meshes in order to function properly. Shadow mapping [1] is another approach that is used widely in games today. With shadow mapping, the scene is first rendered from the point of view of the light, and depth values are stored in a separate buffer. When the scene is then rendered from the normal viewing position, each pixel is transformed again into the light view, and its depth value is compared to the depth value stored in the shadow map. If it is further from the light source, the pixel is in shadow.

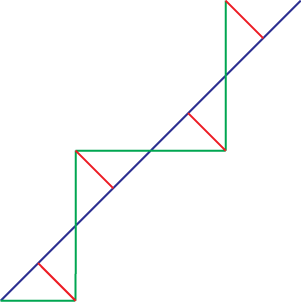
The primary drawback of shadow mapping is that it is an image-space algorithm that is subject to aliasing artifacts due to quantization and perspective projection. Several approaches have attempted to reduce those artifacts: trapezoidal shadow maps (TSM) [2], light-space perspective shadow maps (LiSPSM) [3], perspective shadow maps (PSM) [4], and plane-optimal shadows [5]. All of these approaches suffer from problems that typically prevent their use in games. TSM suffers from “shadow acne” problems (see Section 19.1) and is patented. LiSPSM suffers from significant degradation in its worst case. PSM is highly complex and suffers from singularities that reduce quality. All the above-mentioned approaches do not address nonlinear depth, precluding the use of *z* bias to remove shadow acne, with one notable exception from Kozlov [6].

This gem offers a complementary explanation of a different approach to shadow mapping that was first proposed by Gussev [7]. It is recommended that the reader consult this paper after reading this gem for further information. We assume that the reader already has a good understanding of the subject material and does not go in very much detail about the background principles. In short, the primary effect that you want in your shadow buffer is that shadows close to the camera utilize more shadow map texture area than shadows farther away. eXtended Perspective Shadow Mapping (XPSM) accomplishes this by first finding a projection vector in light space and generating a matrix to warp the space according to that vector. It then generates an affine transform that moves the warped space into normalized device coordinates that does not affect the warping property.

**19.1 General Overview**

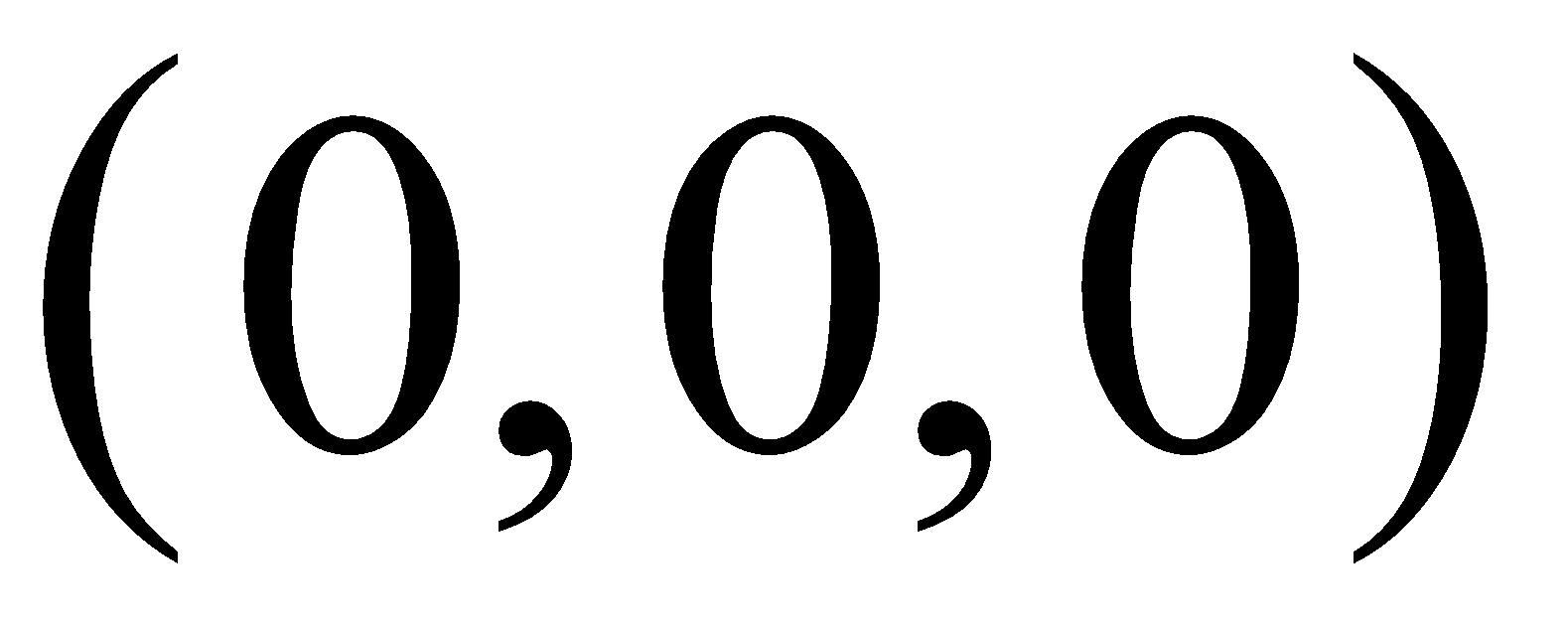
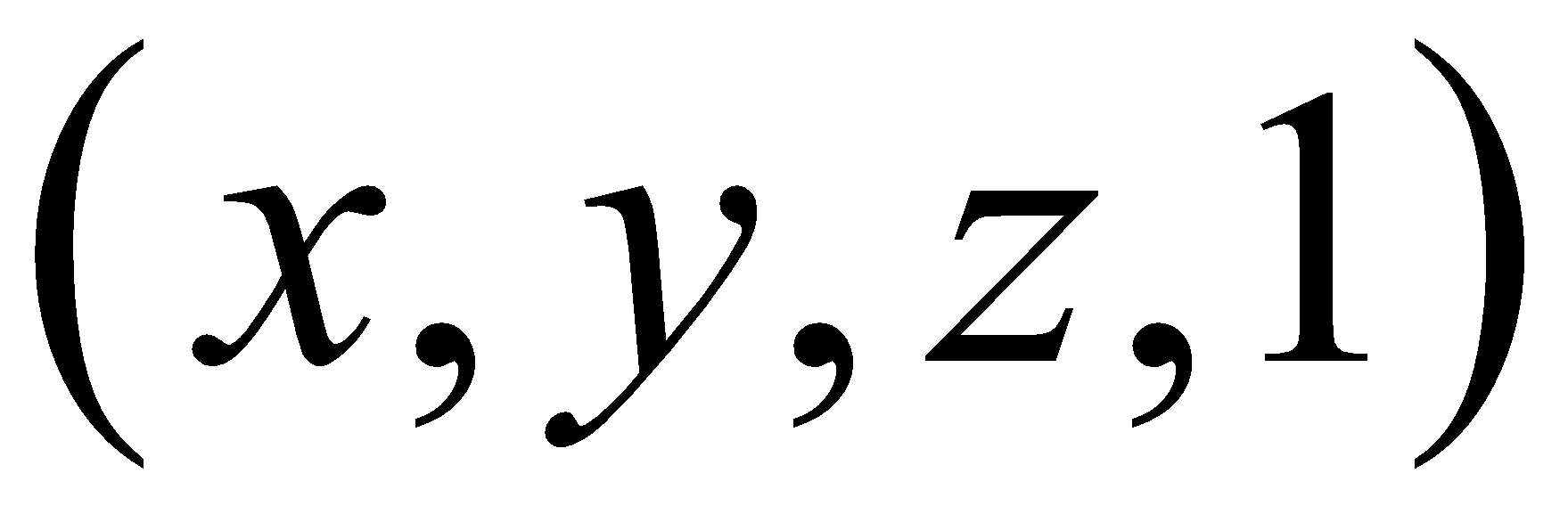
**What is shadow acne?**

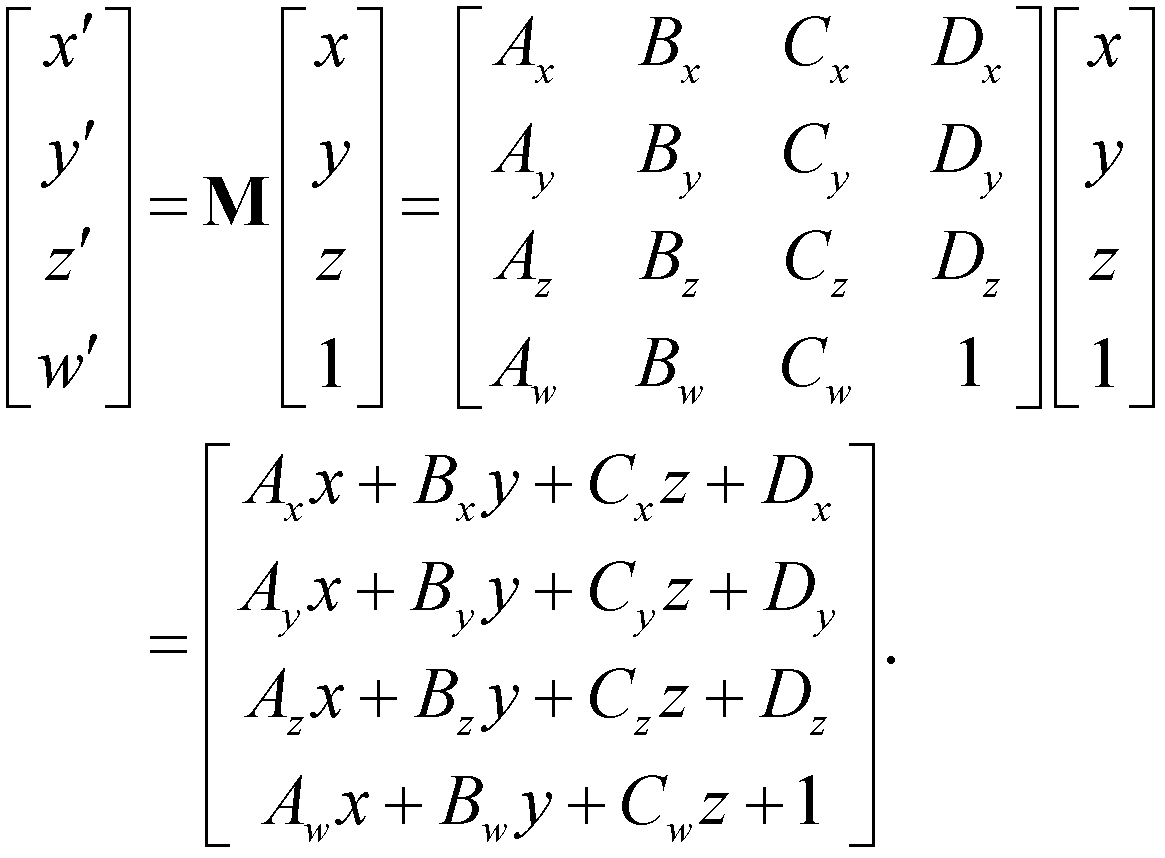
A typical problem with shadow mapping solutions is shadow acne. This occurs because the shadow map pixels do not directly coincide with primary view pixels. More specifically, the shadow map samples triangles at discrete points in space, creating a stair step, whereas the actual surface is continuous. The main view samples that same surface at different discrete points. Shadow acne occurs when those two discrete stair steps do not exactly coincide, resulting in some samples incorrectly being in shadow or not in shadow. This problem is typically solved by using a *z* bias to shift the shadow map samples closer to or further away from the light. With perspective shadow mapping, however, *z* bias has nonlinear depth issues that we cover for XPSM later in this gem.

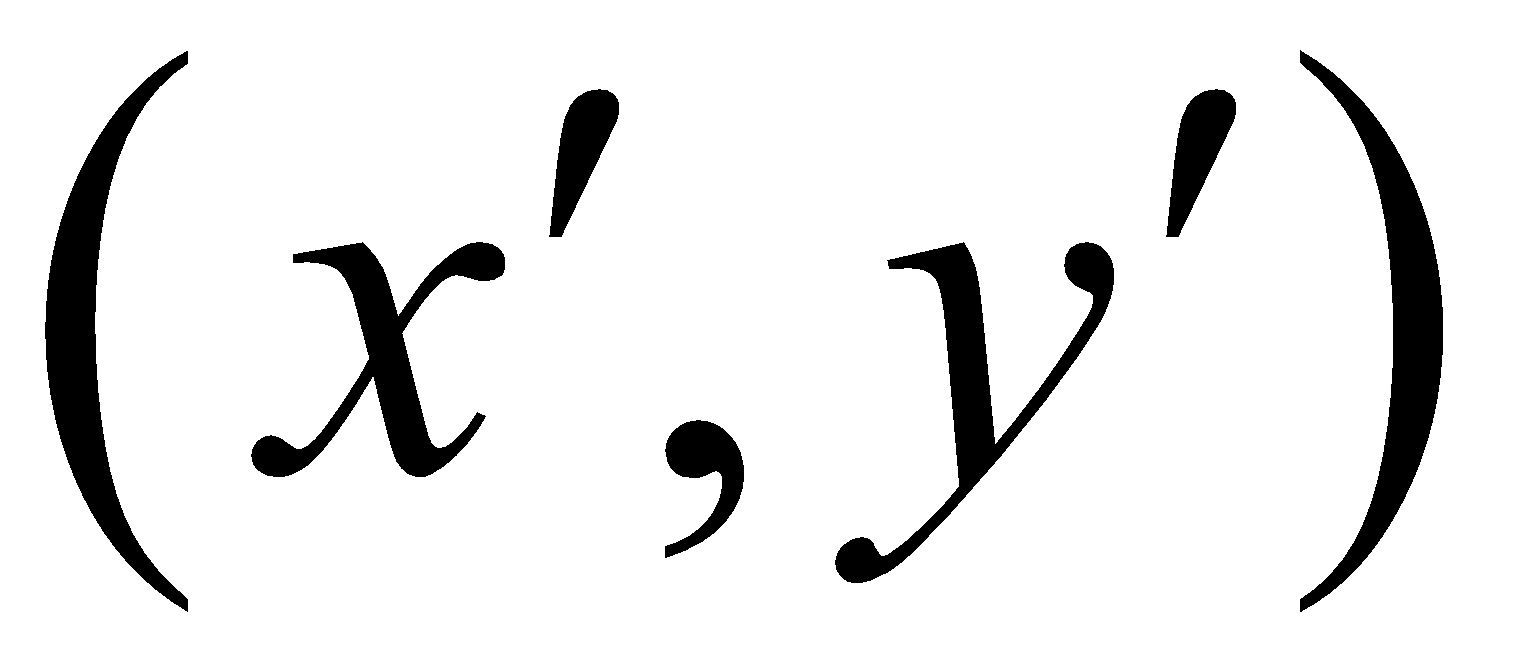


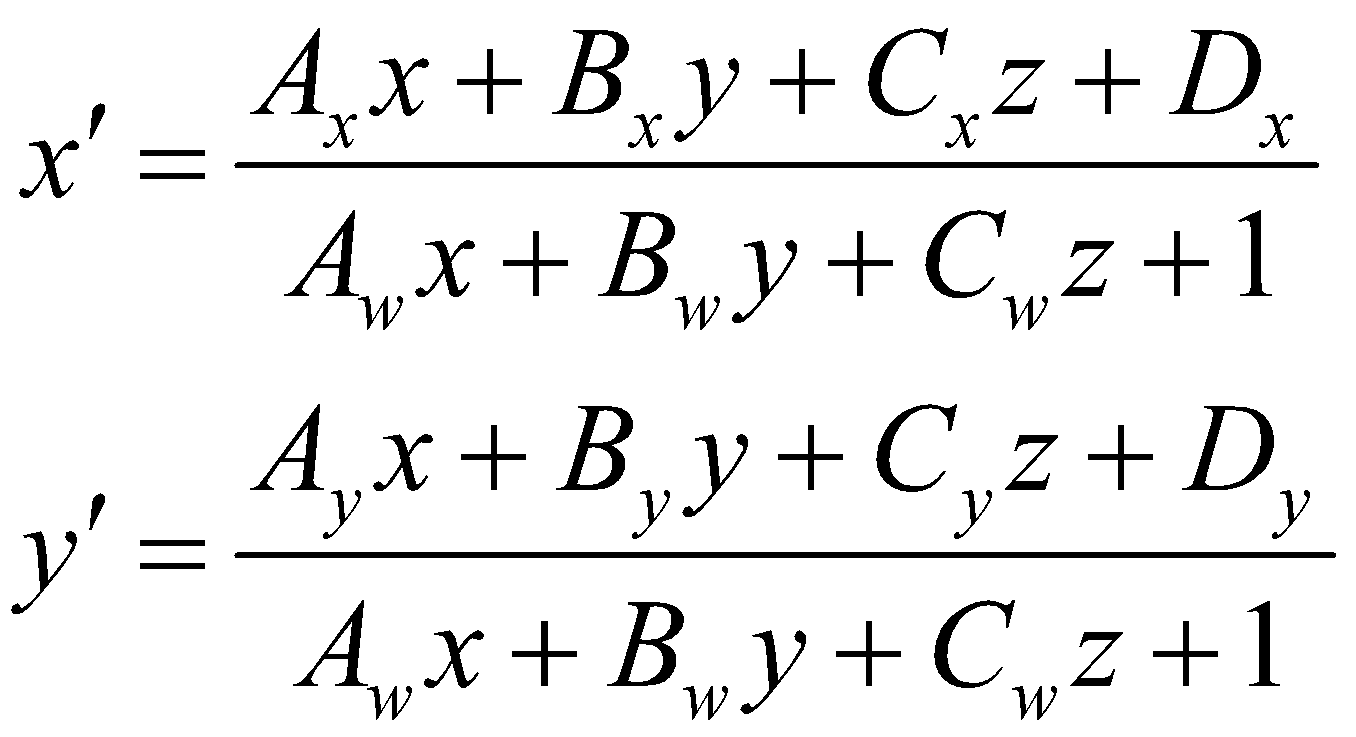
**Figure 19.1.** The blue line represents a triangle surface, and the green lines represent the shadow map discrete sample locations. The red lines represent the differences between the shadow map depth values and the actual surface. It is this difference that causes shadow acne.

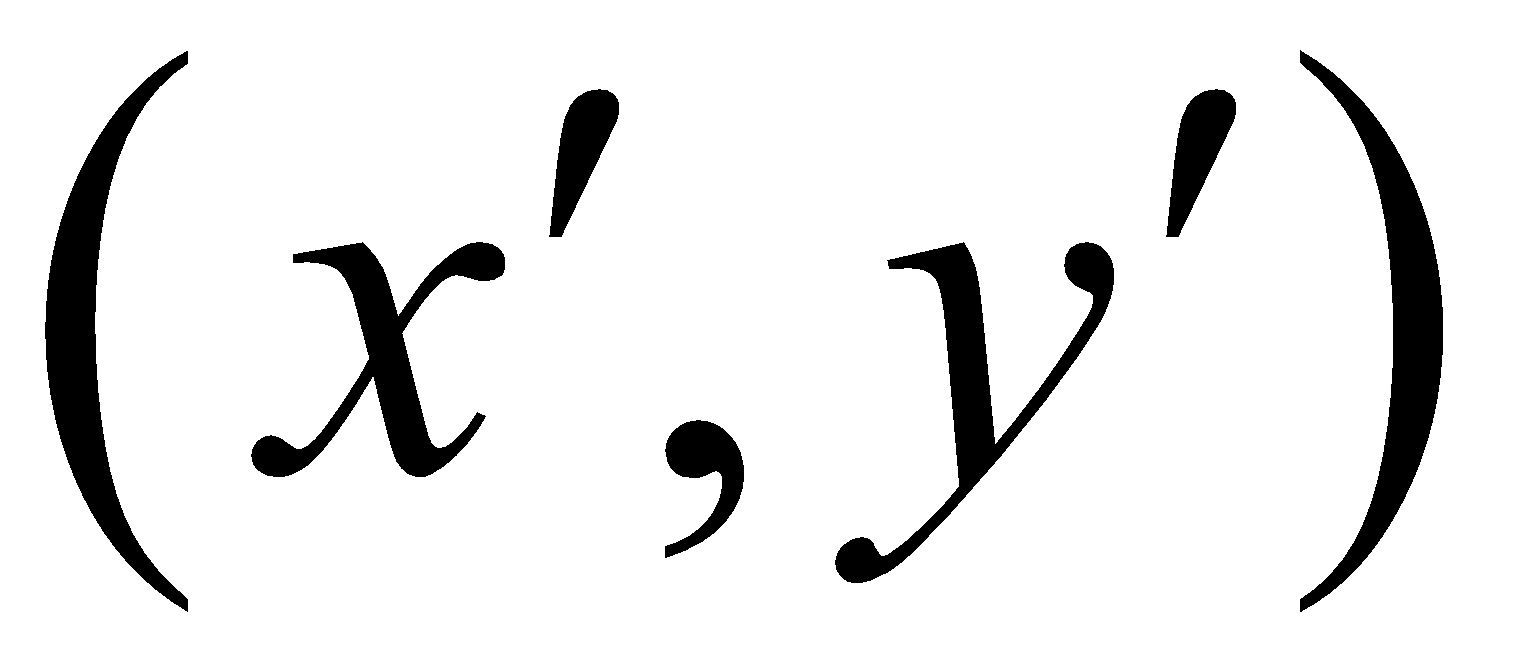
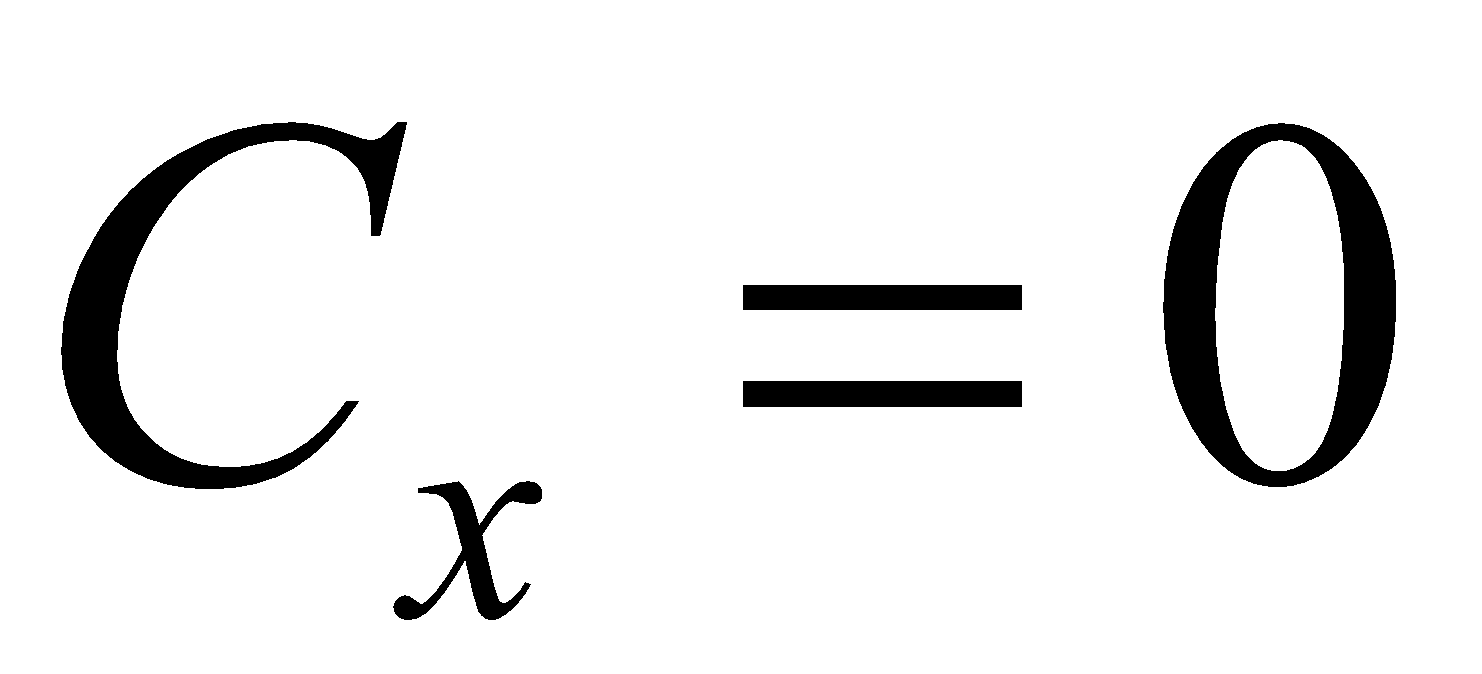
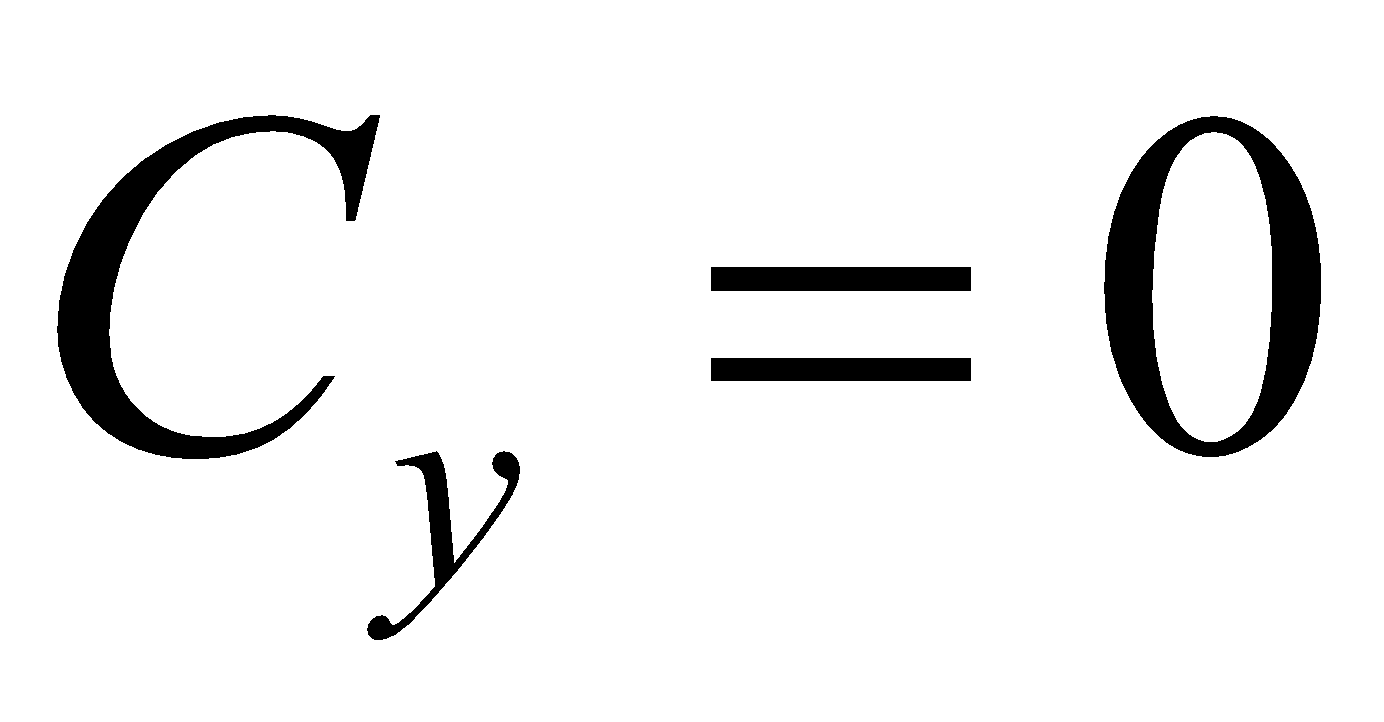
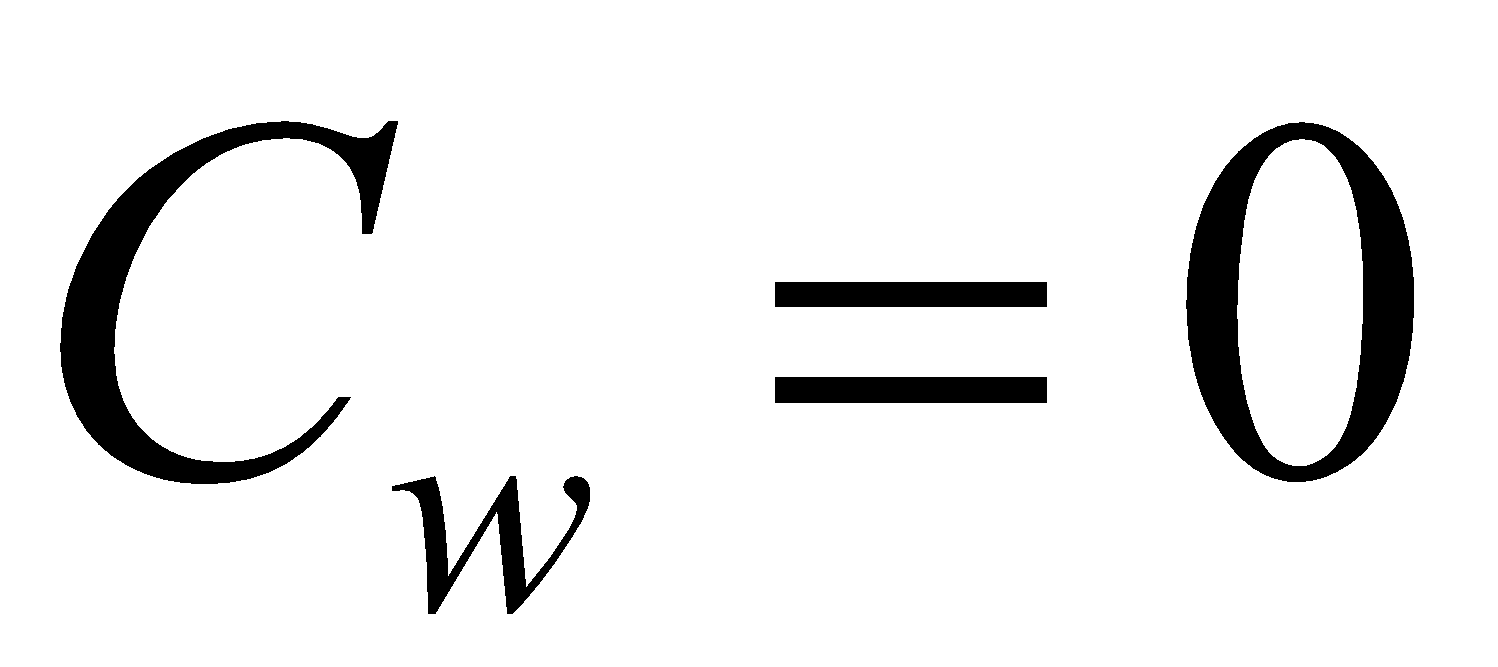
**Parameterization plane**

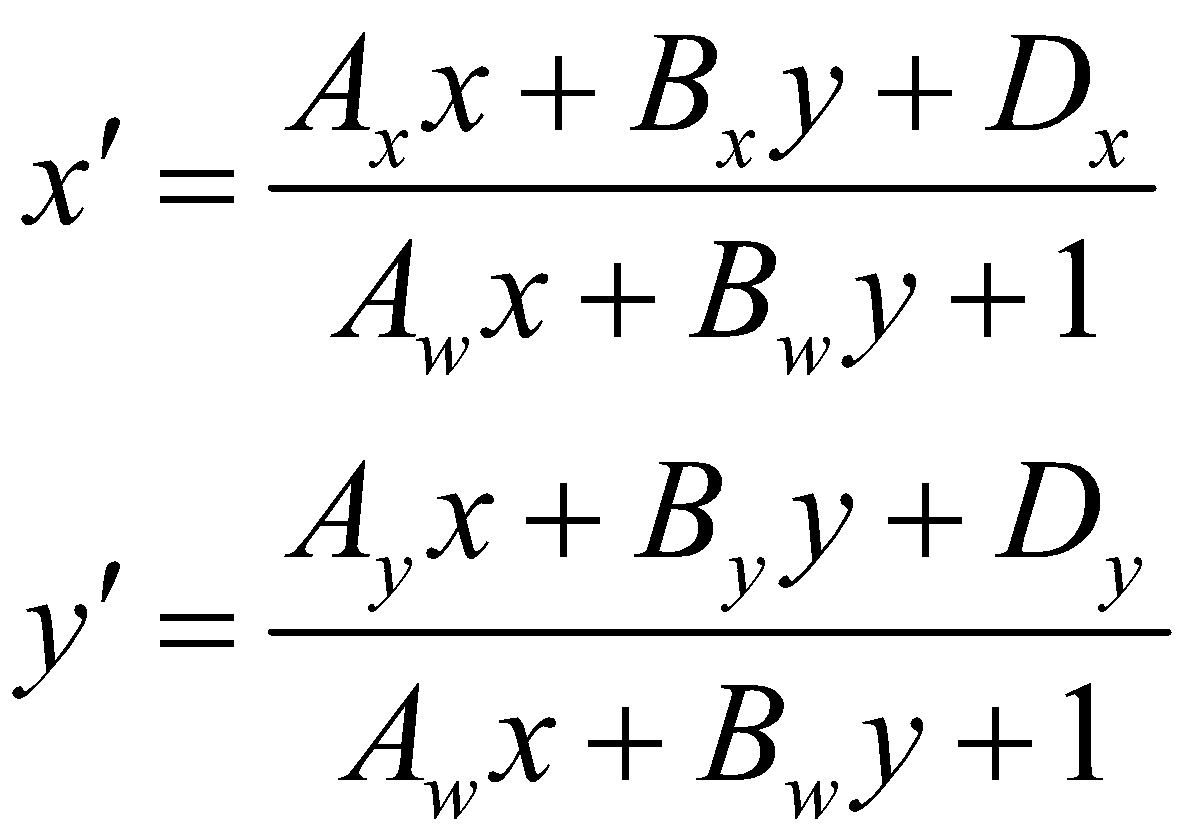
In light space, the light direction is parallel to the *z-*axis, and the light origin is at . The projection of a point  in light space by a projection matrix **M** is given by

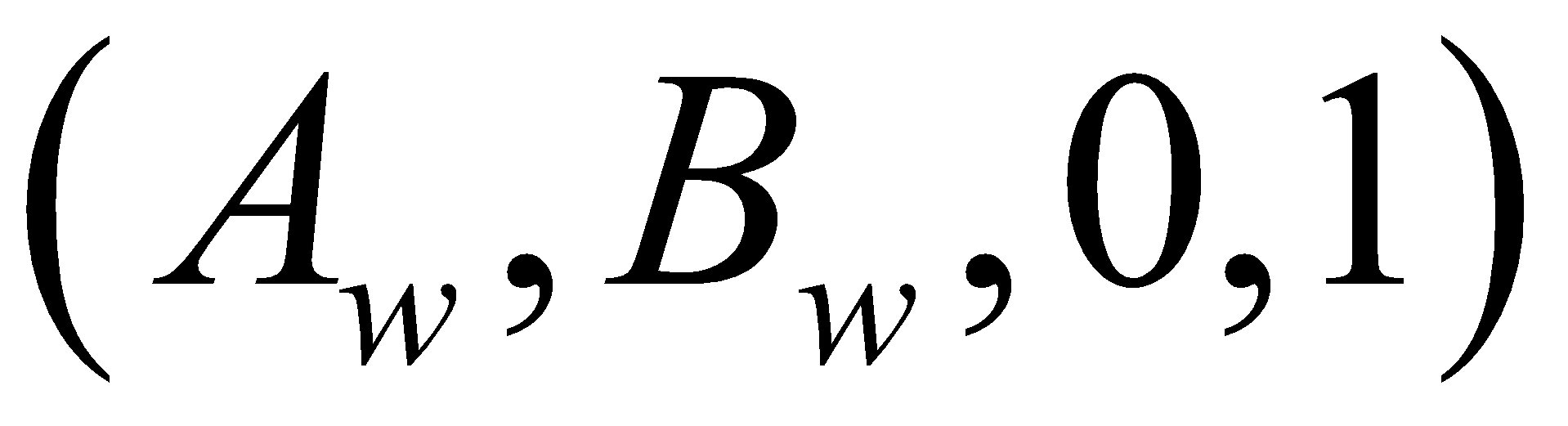
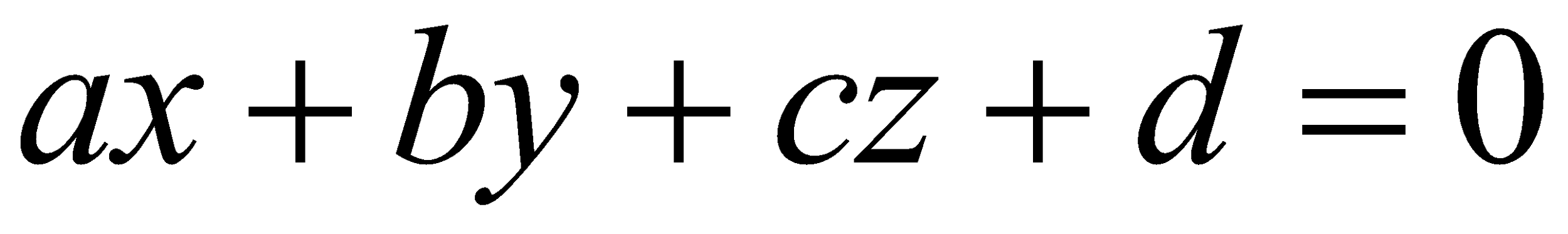
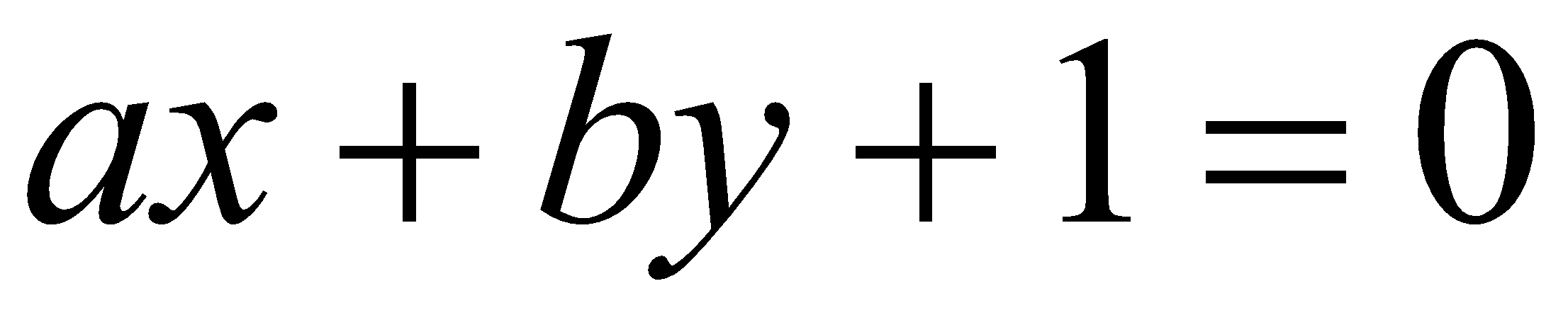
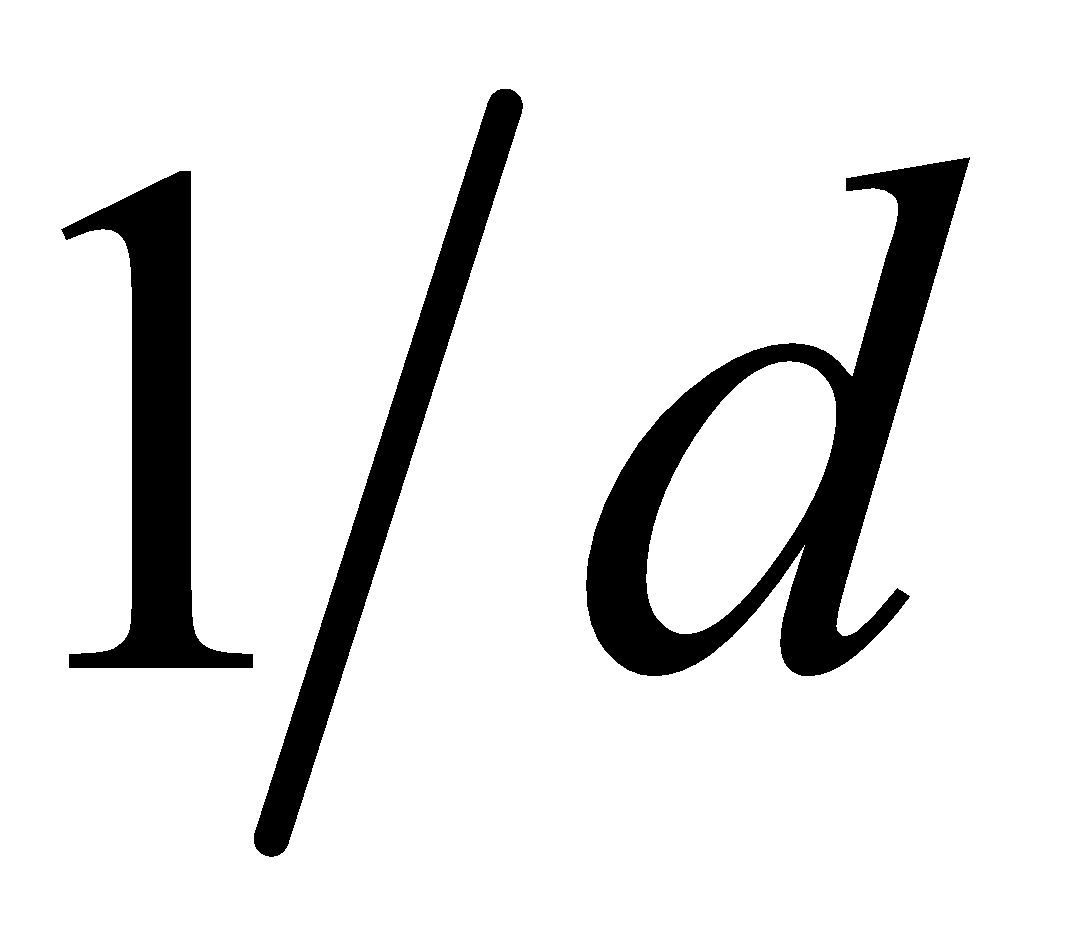
(19.1)

After division by the resulting *w*-coordinate, the projected *x*- and *y*-coordinates  can be expressed as

. (19.2)

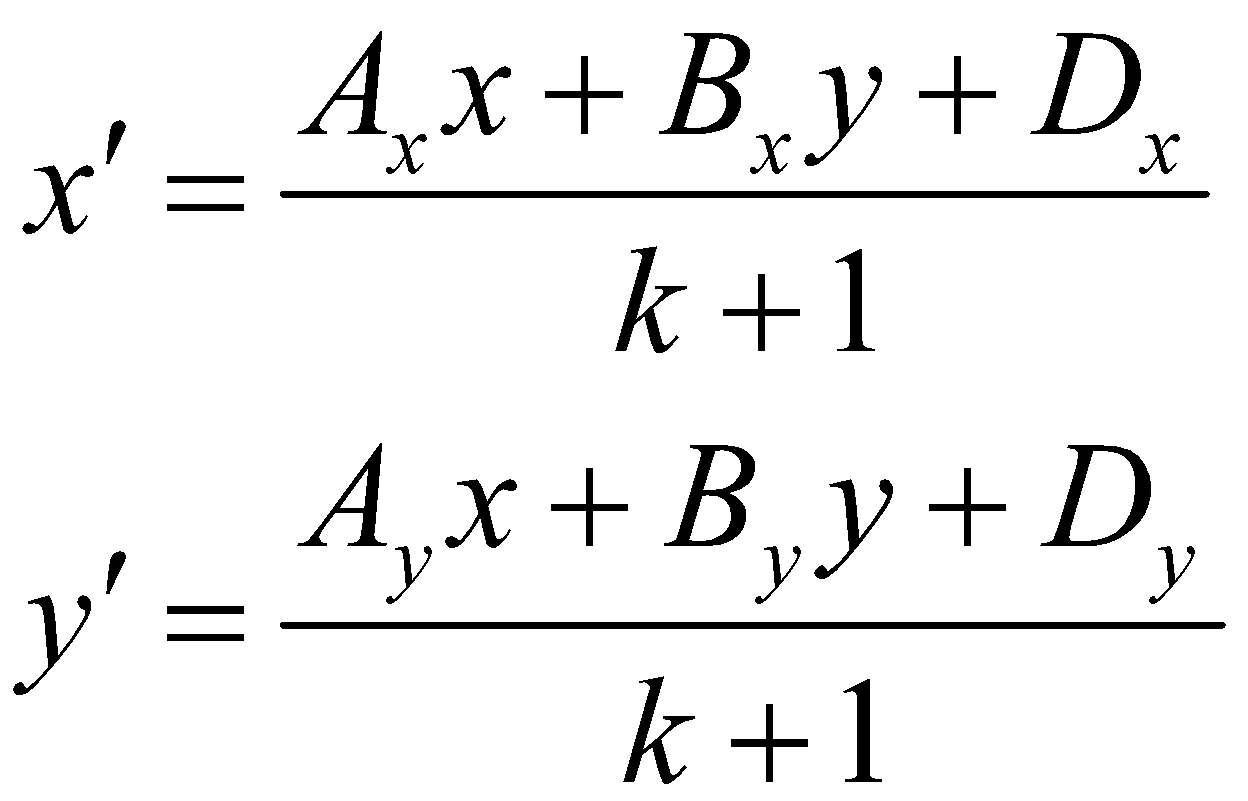
Any light ray must project to the same  point on the shadow map no matter what the position *z* along the ray. In order for this property to be satisfied, we must have the constraints , , and . Equation (19.2) then simplifies to

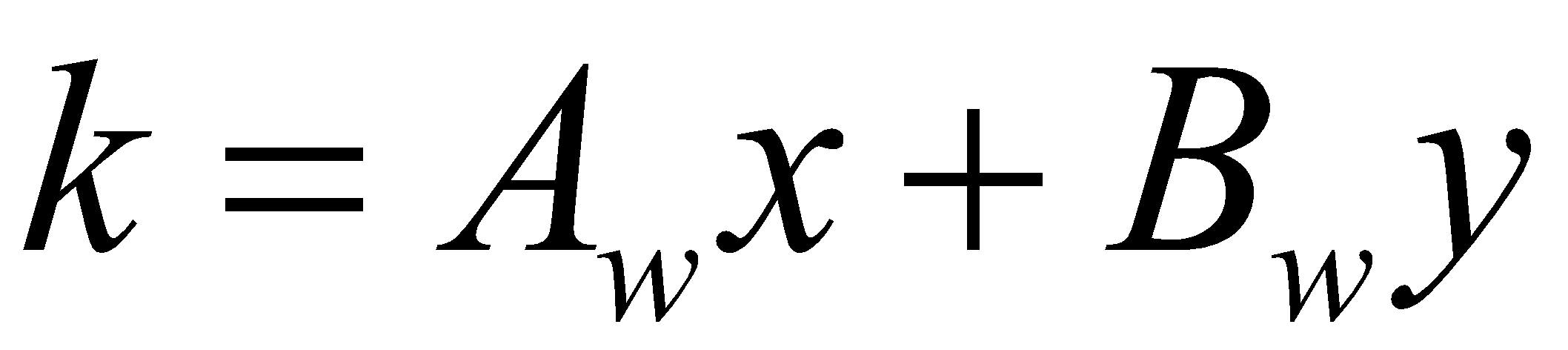
. (19.3)

The vector  appearing in the denominator is called the *parameterization vector*. It’s direction and length constitute the *parameterization plane*. For example, a typical 2D plane equation is . In this equation for our specific purpose, *c* is equal to 0, and *d* is fixed to 1, simplifying the equation to . In order to be able to fix *d* to 1, *a* and *b* must be scaled by  in order to accommodate. Armed with this, we can generate a plane to expand the shadow texture space closer to the view and squish the shadow texture space further away.

**Back projection singularity**

Equation (19.3) is equivalent to

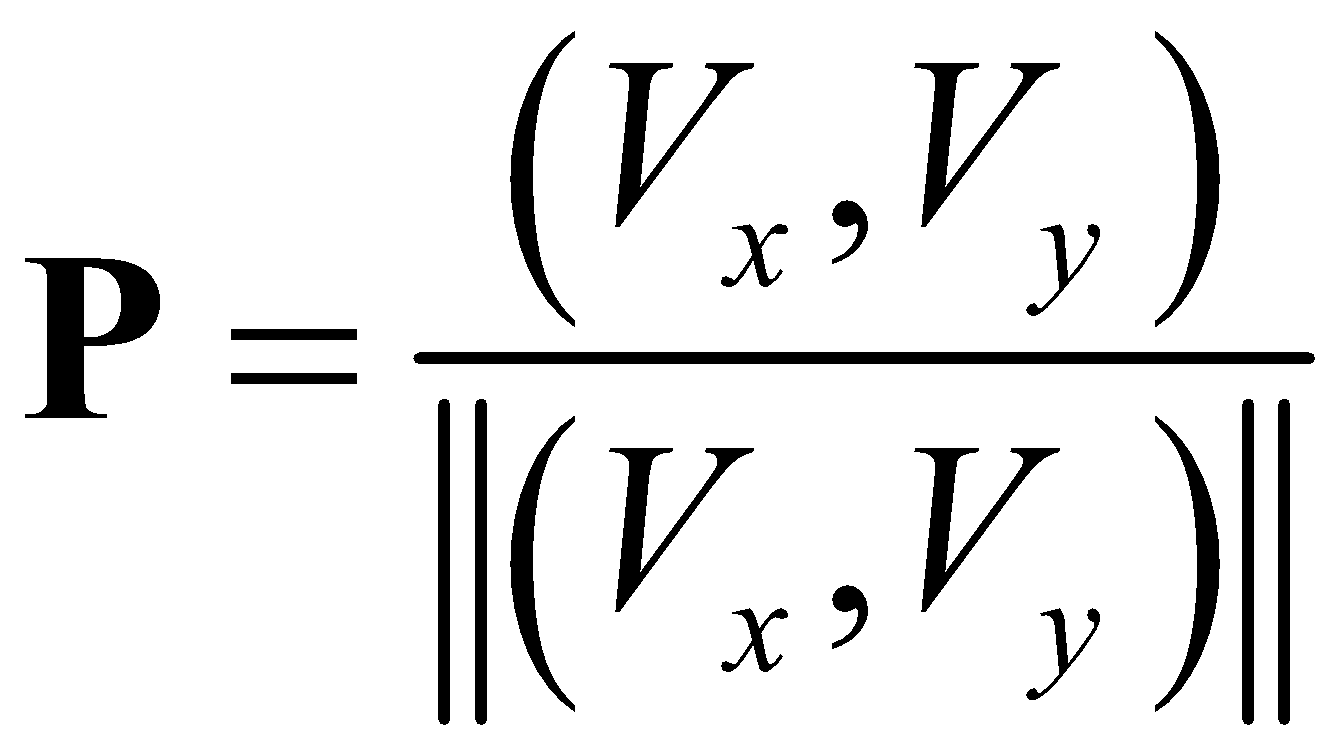
, (19.4)

where . There is a singularity when , occurring when shadow casters lie far behind the viewer. In this case, the only solution is to move the parameterization plane behind these points using a very small parameterization vector, as is discussed in the next section. This compression of space affects the view frustum, causing significant degradation in shadow quality. This problem exists in all perspective shadow mapping solutions.

**19.2 Implementation Details**

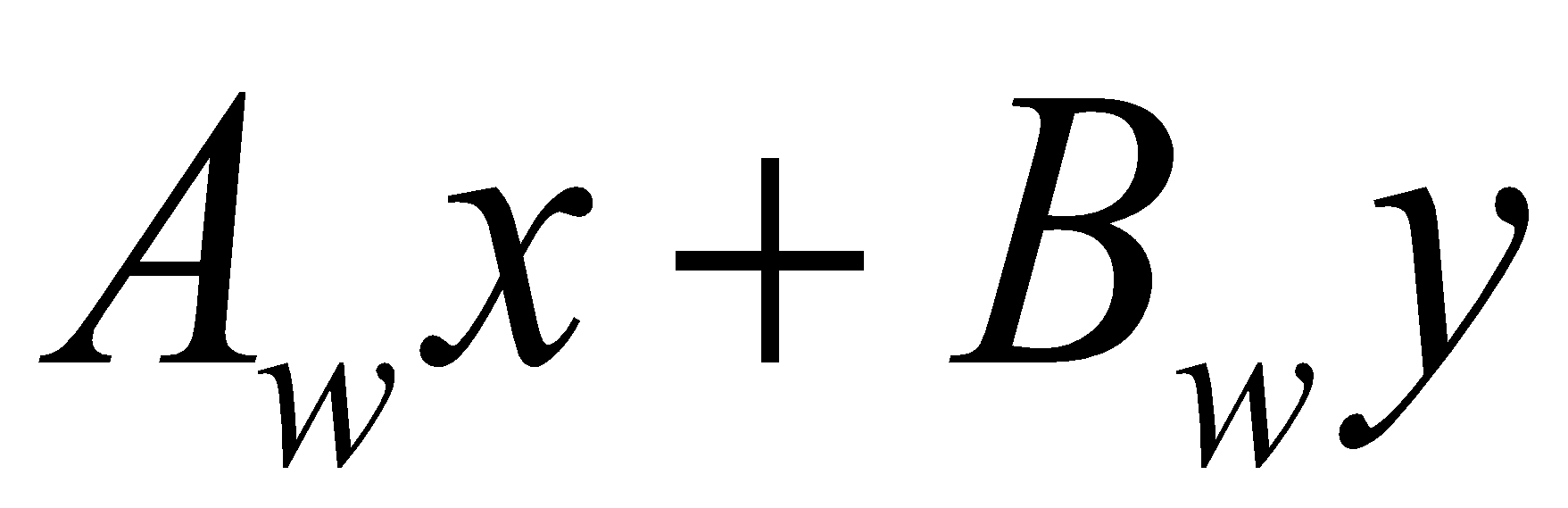
**Projection vector direction**

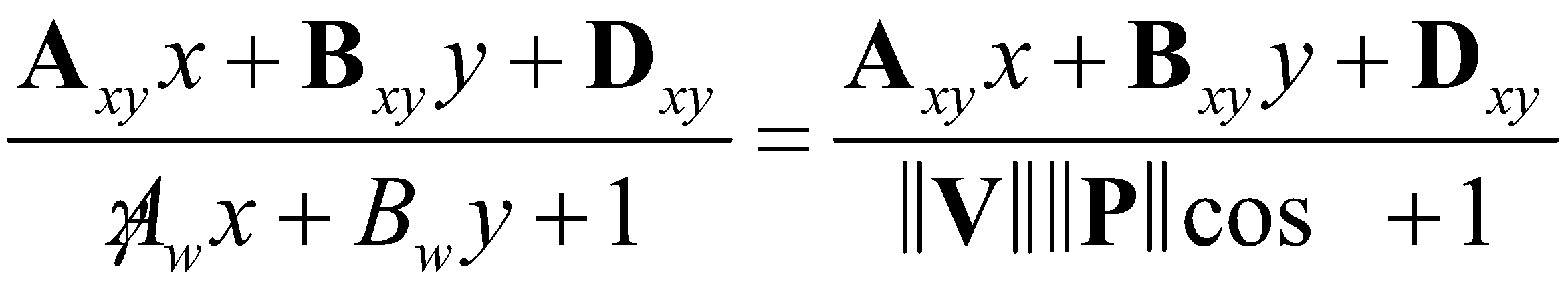
The depth buffer for the primary camera is hard to predict ahead of time and can be random in the worst case. For example, it is hard to tell in advance when the bottom of the screen will have more pixels closer to the camera than the top. If we knew that, then the parameterization plane could be set up to specifically take advantage of it. It is theoretically possible to choose an optimal parameterization plane based on previous frame’s depth information; however, a generally good choice for the common case is a symmetrically distributed projection vector for the entire view frustum. This line of symmetry is equal to the view direction **V** in light space projected onto the 2D *x*-*y* plane. Thus, the direction of the parameterization vector is equal to

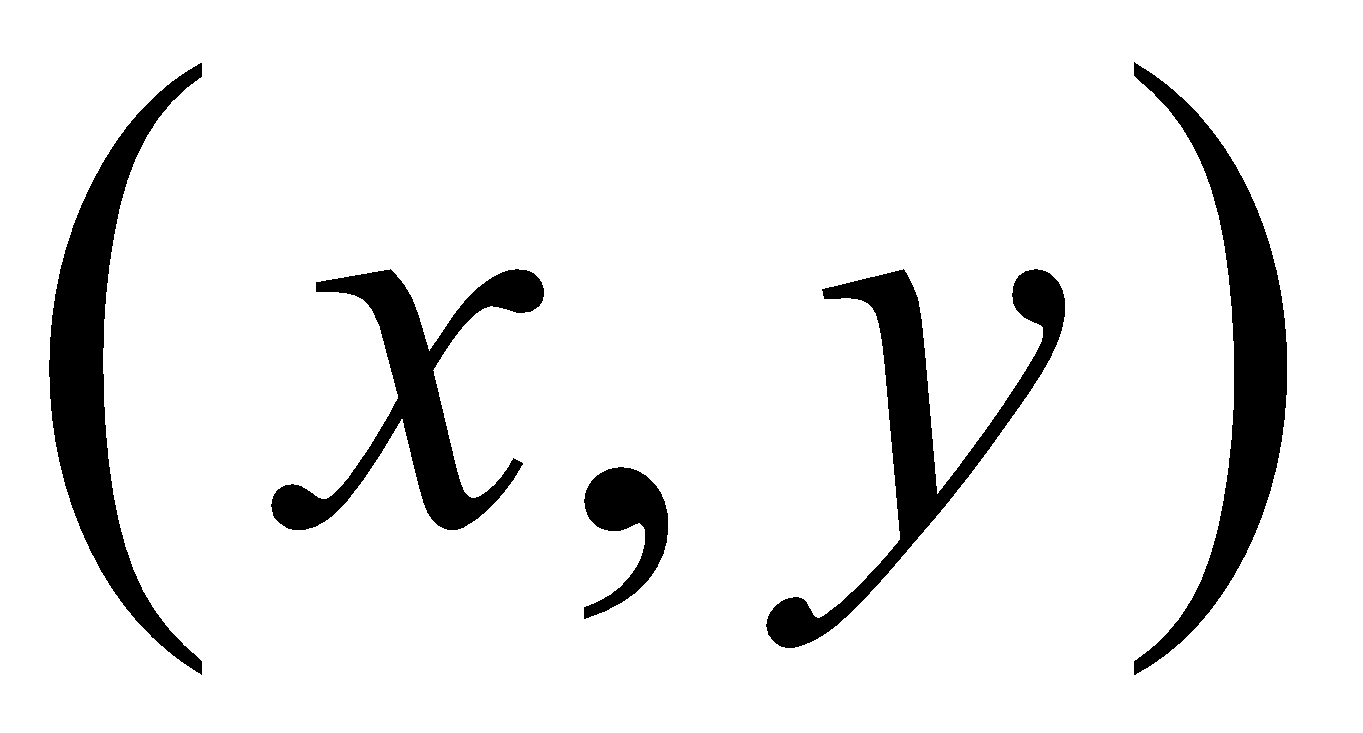
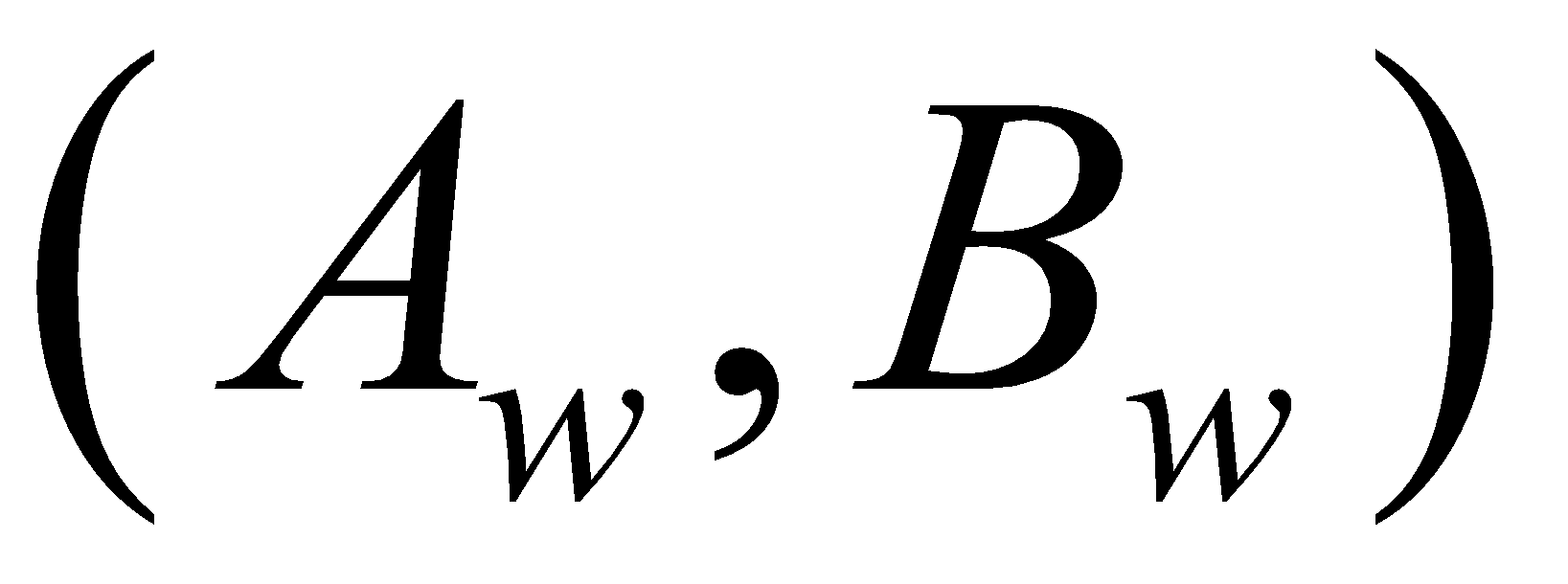
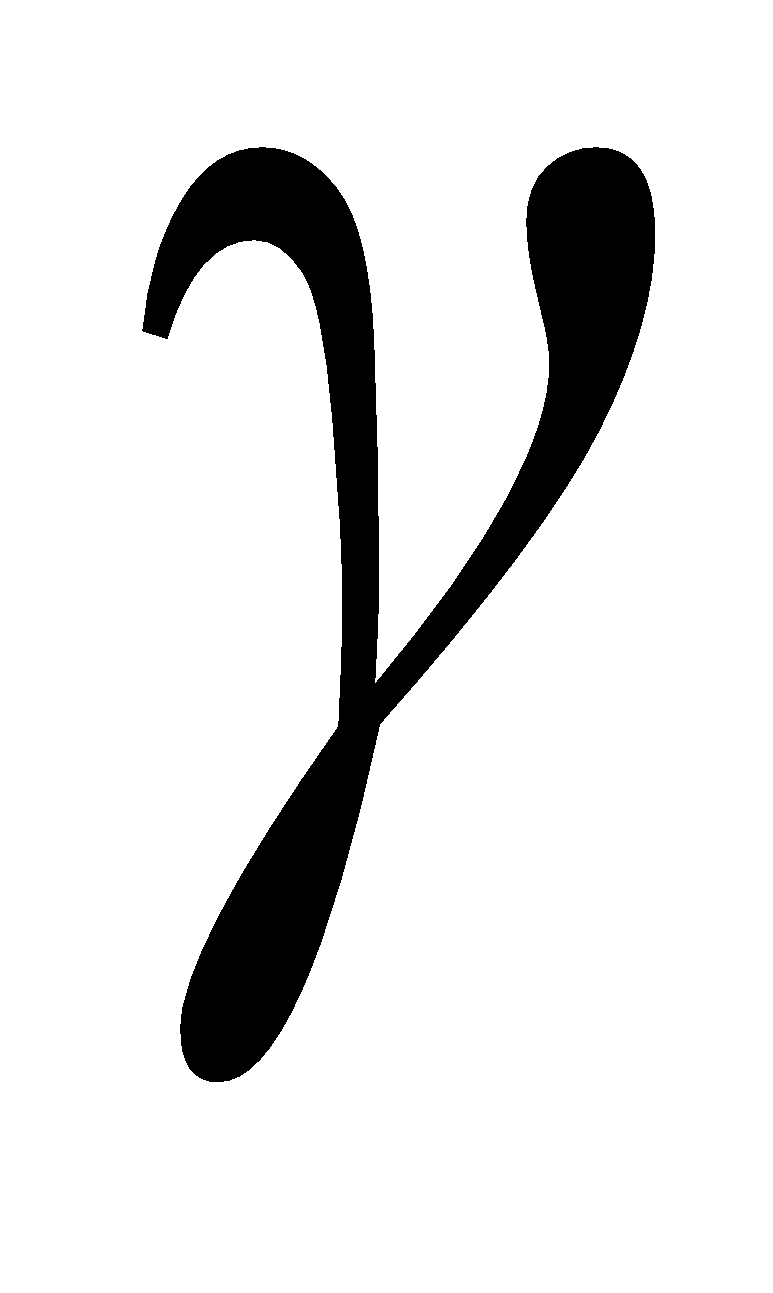
. (19.5)

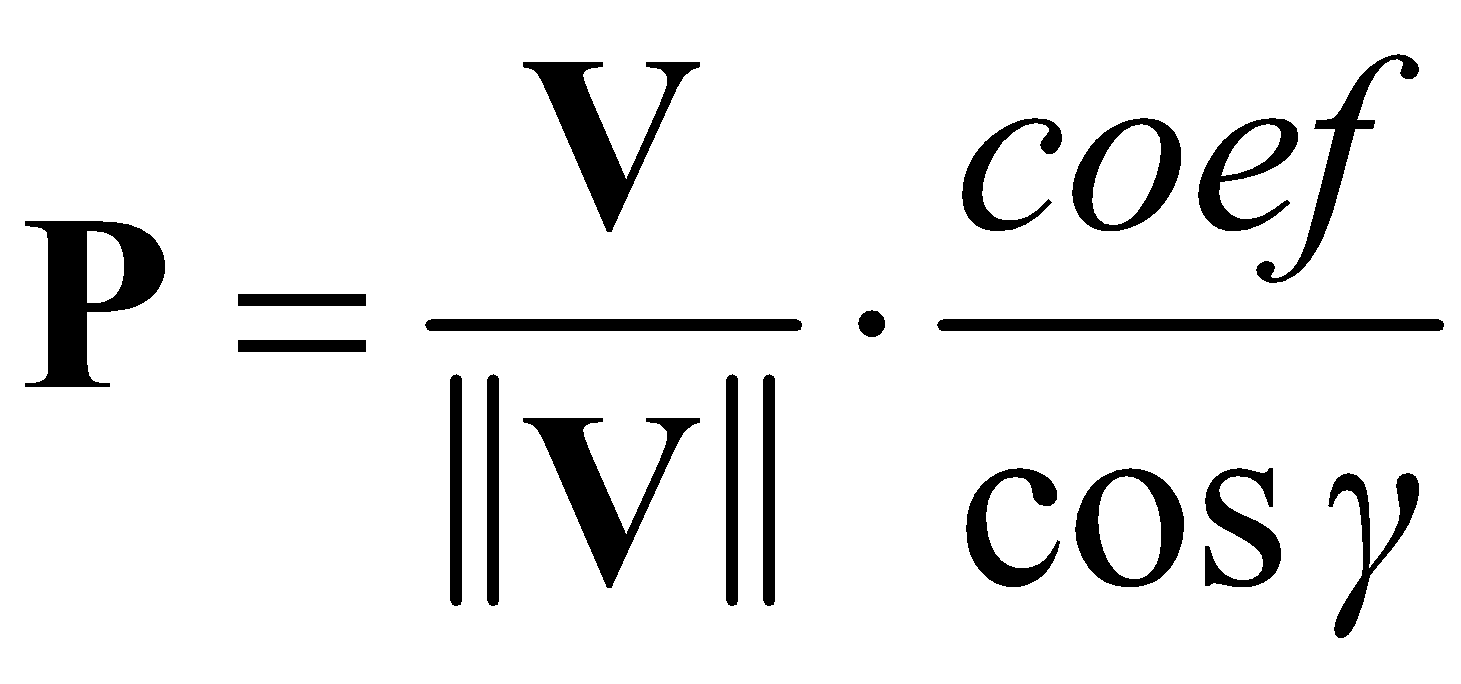
**Optimal warping**

Ideally, the warping effect would only depend on the distance from the view location. That is, near objects have higher shadow map resolution and farther objects have lower shadow map resolution. The shadow map resolution should not depend on the relative viewer and light orientations. The key observation here is that orthogonal shadow mapping is good for far away shadows and perspective shadow mapping is good for near shadows. What we seek to find is a knob that we can turn to change a perspective solution into an orthogonal one and find that magic balance between the two.

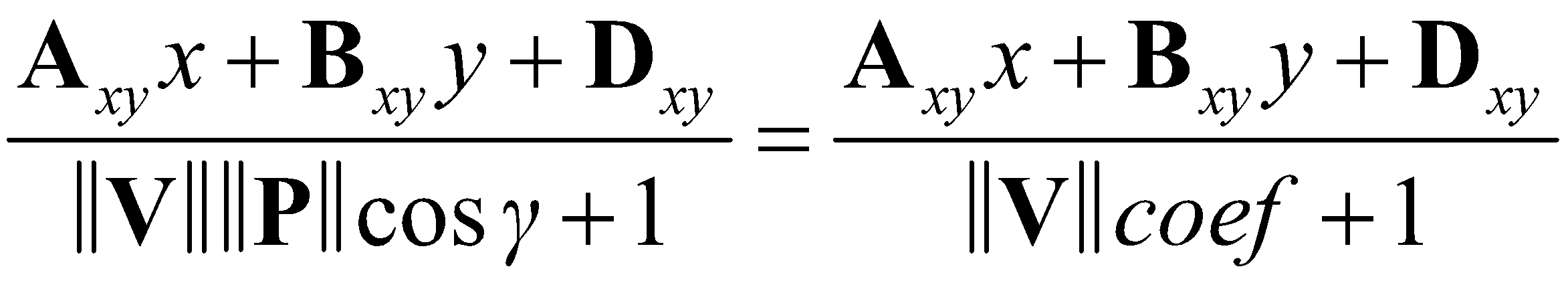
To start, recognize that  is a two-dimensional dot product, and as such, we have the identity

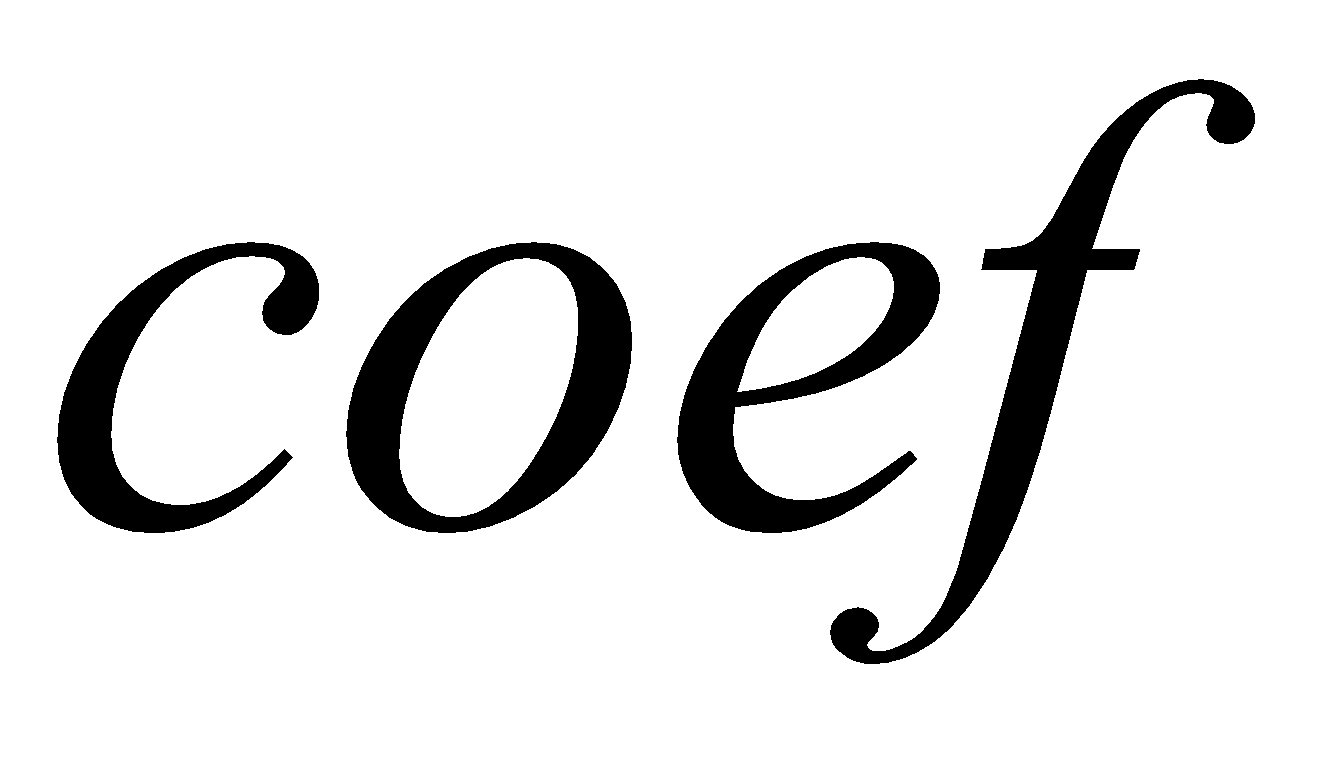
, (19.6)

where **V** is a view vector , **P** is the projection vector , and  is the angle between them. If the desired effect is to increase or decrease the warping effect, then **P** can be defined as

. (19.7)

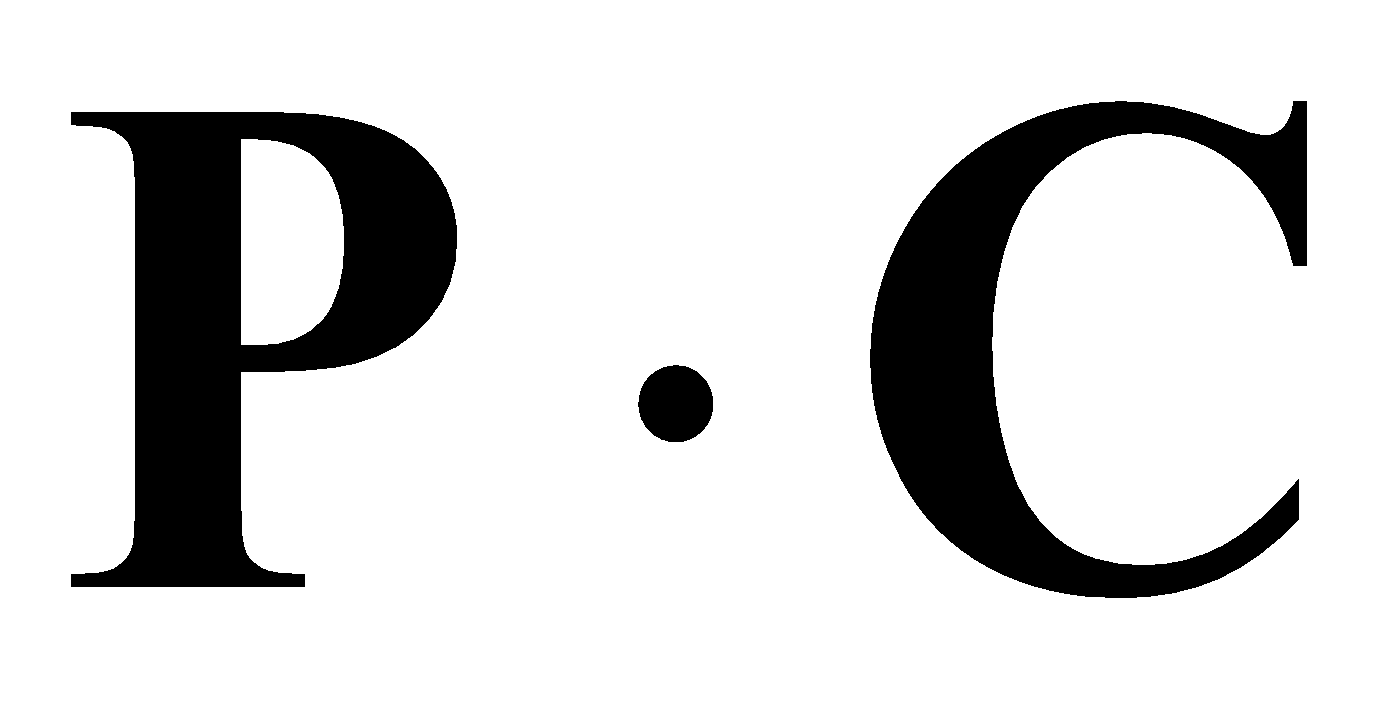
Then Equation (19.6) becomes

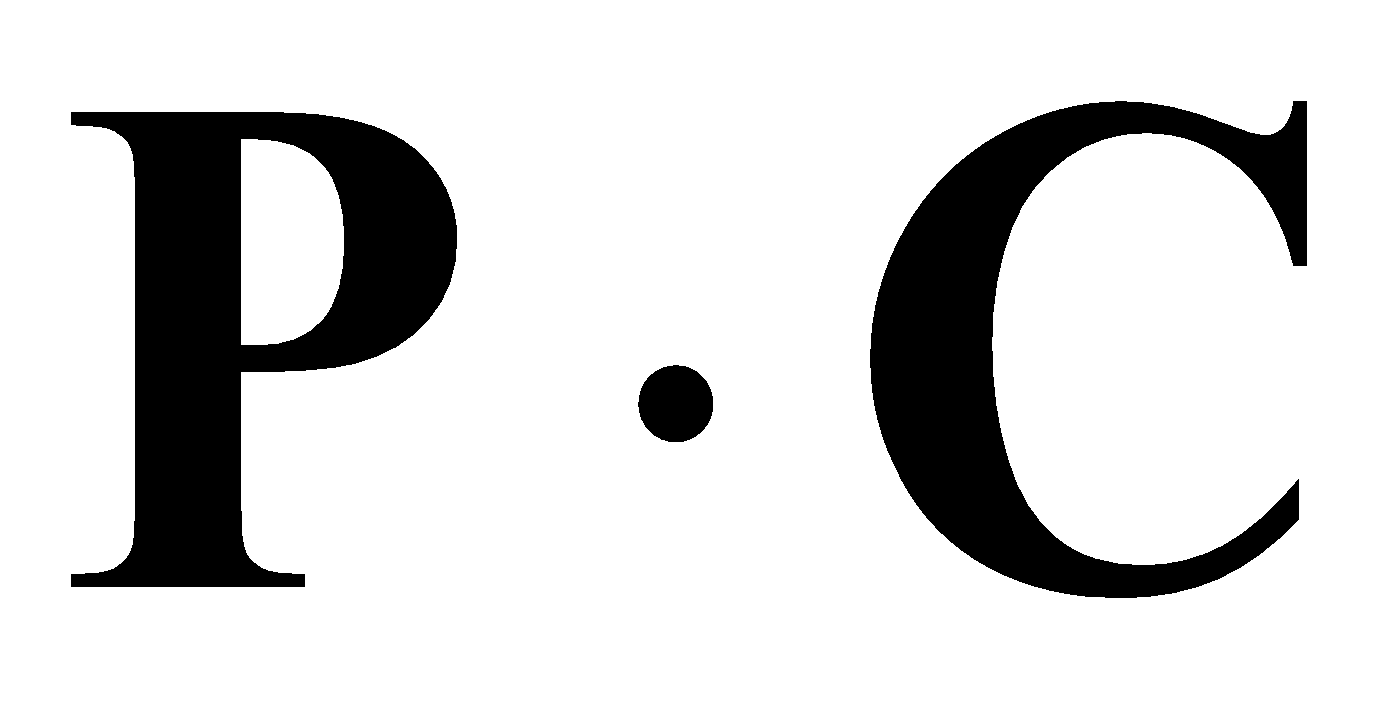
, (19.8)

where  is tuned for scene scale and affects the trade-off between near shadow resolution and far shadow resolution.

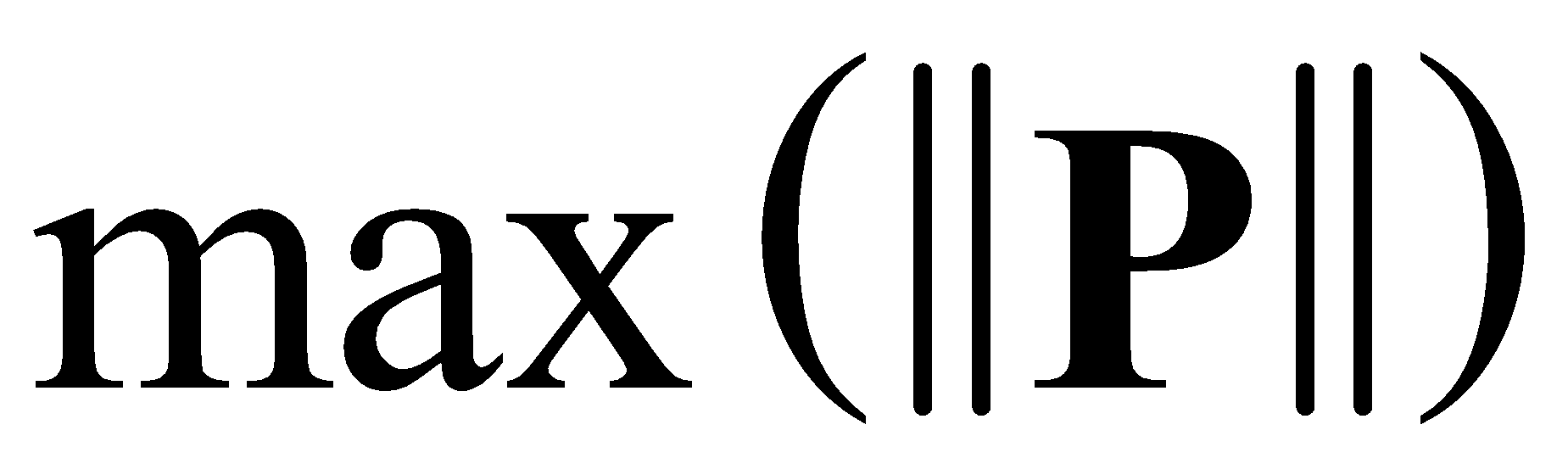
**Back projection singularity**

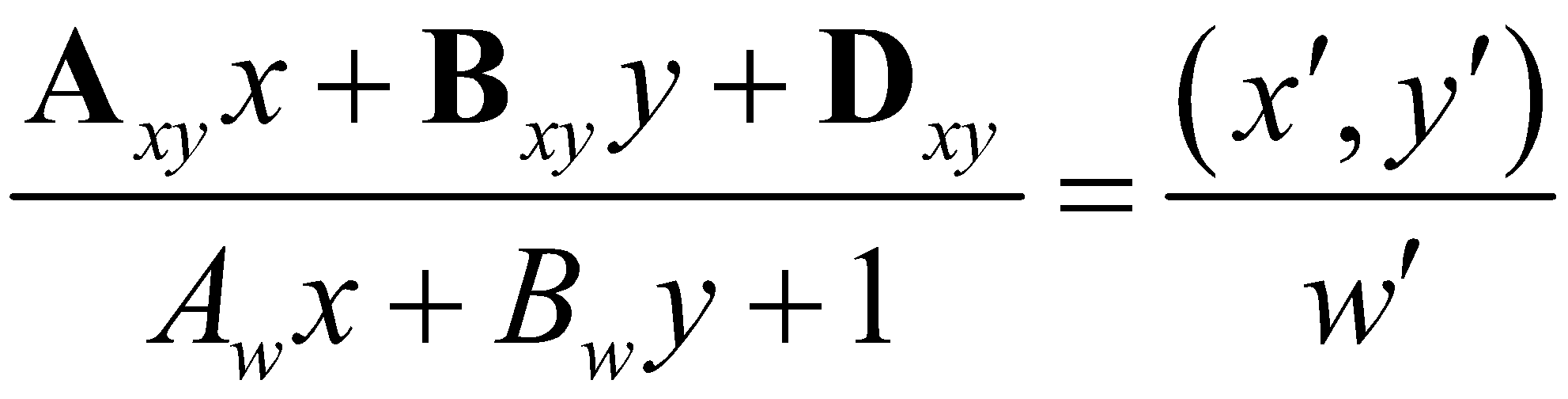
To properly handle the back projection singularity, all casters behind the camera must lie within the unit cube. This can be handled by computing the bounds of the parameterization vector length.

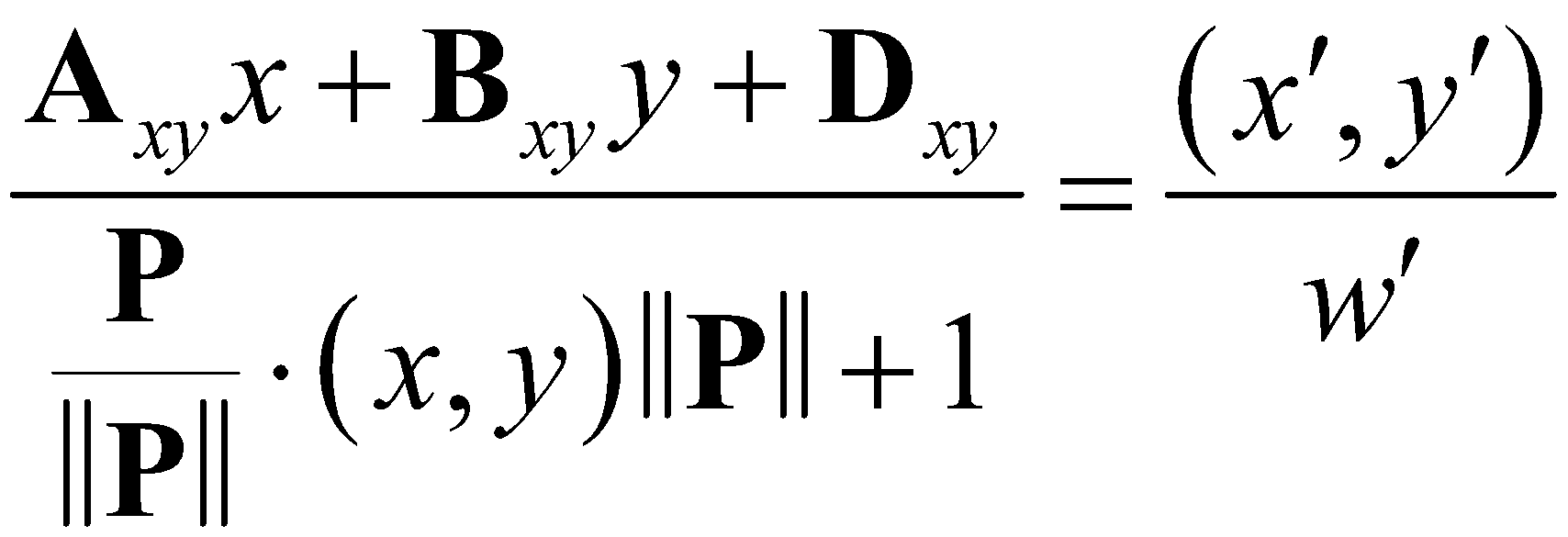
◼ For all shadow casters, find the minimum projection of the caster extents to **P**. This is, find the minimum  for all **C**, where **C** is any location of a shadow castor.

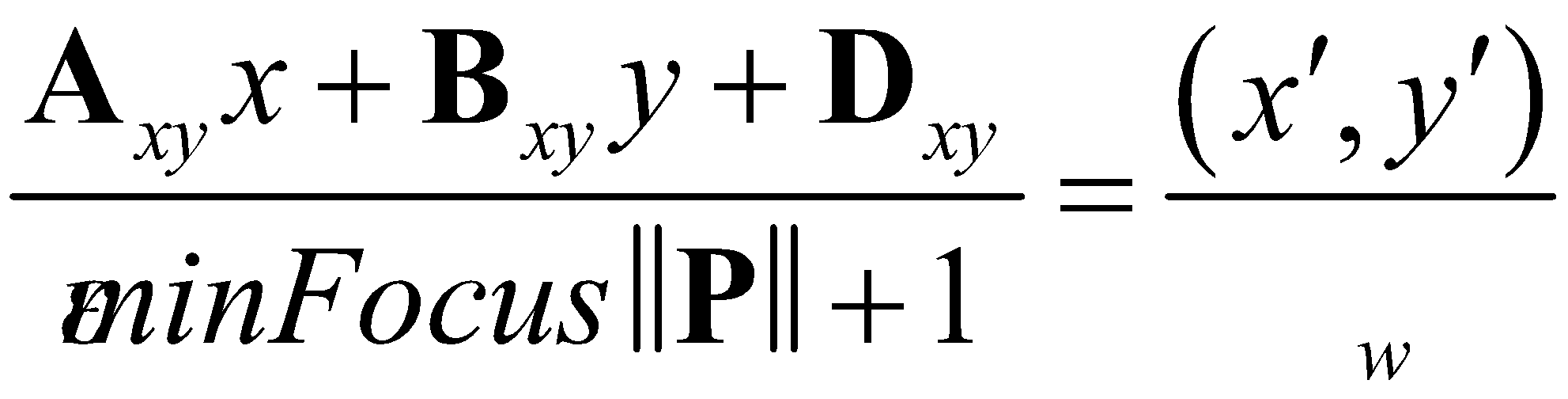
◼ For all shadow receivers, find the minimum projection of the castor extents to **P**. That is, find the minimum  for all **C**, where **C** is any location of a shadow receiver.

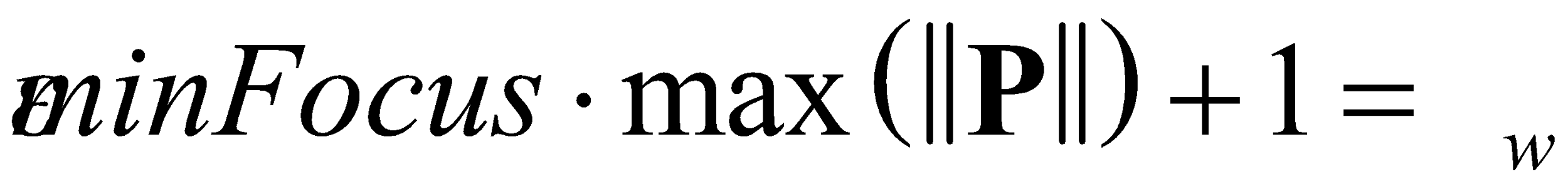
The focus region minimum is defined as the intersection of the minimum projection of shadow castors and minimum projection of shadow receivers.

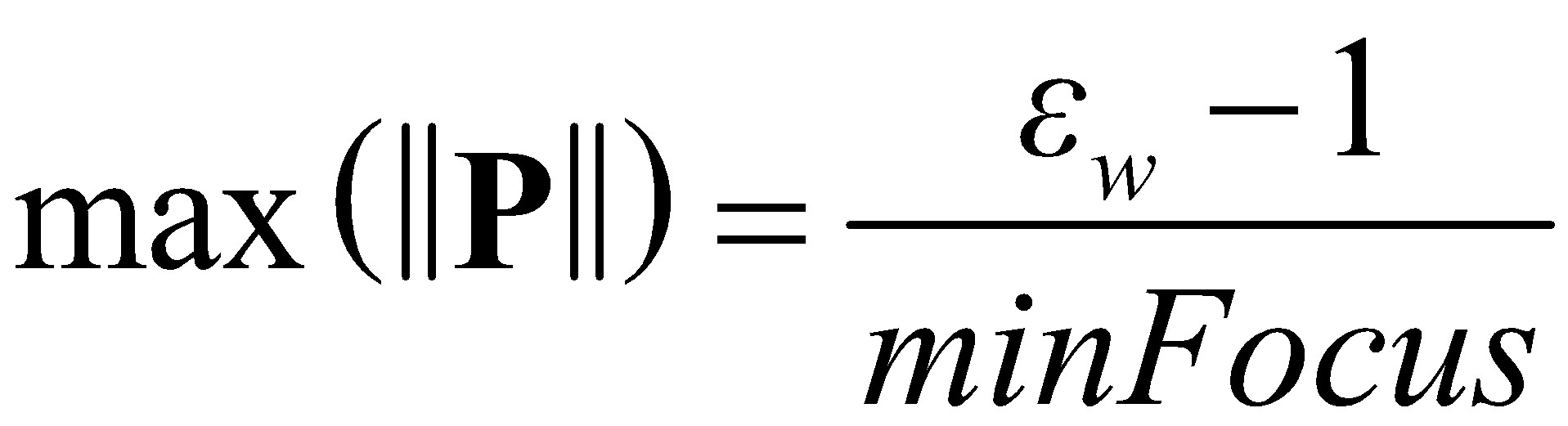
To calculate the maximum parameterization vector length , start by referring to Equation (19.6):

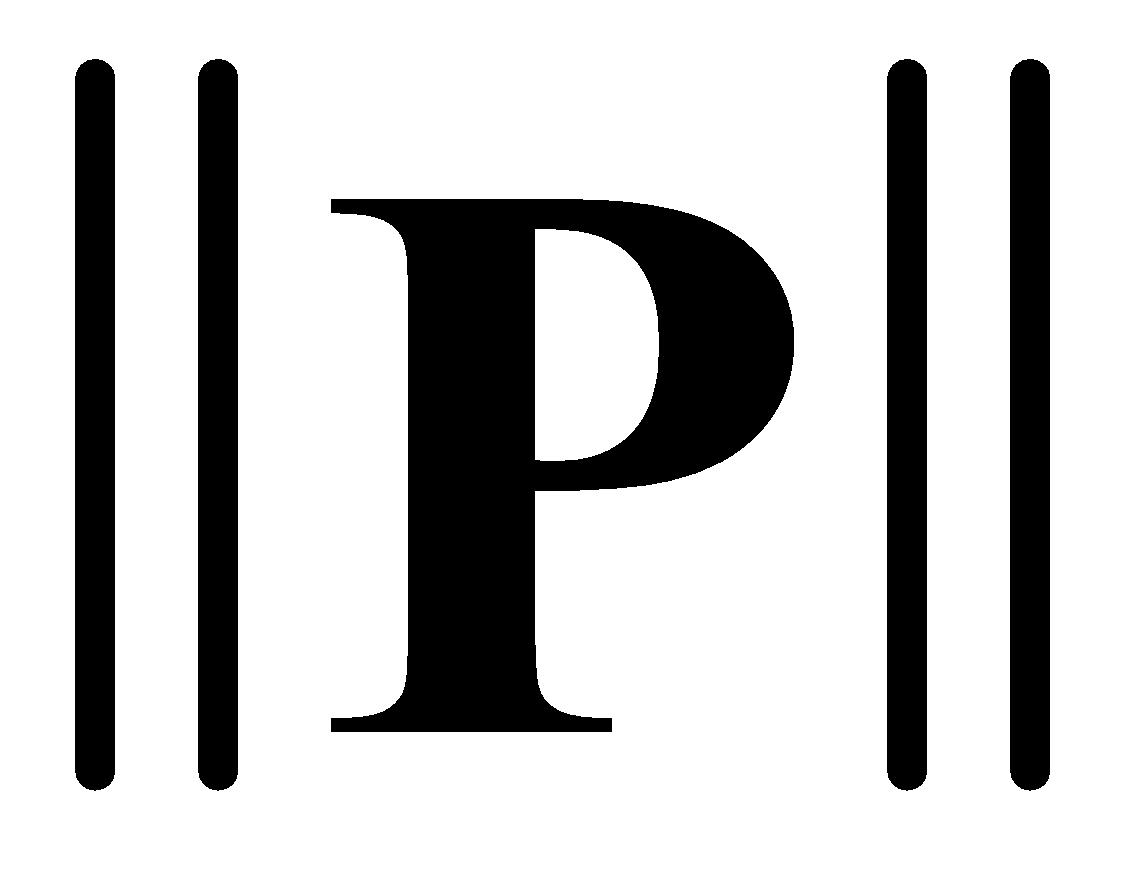
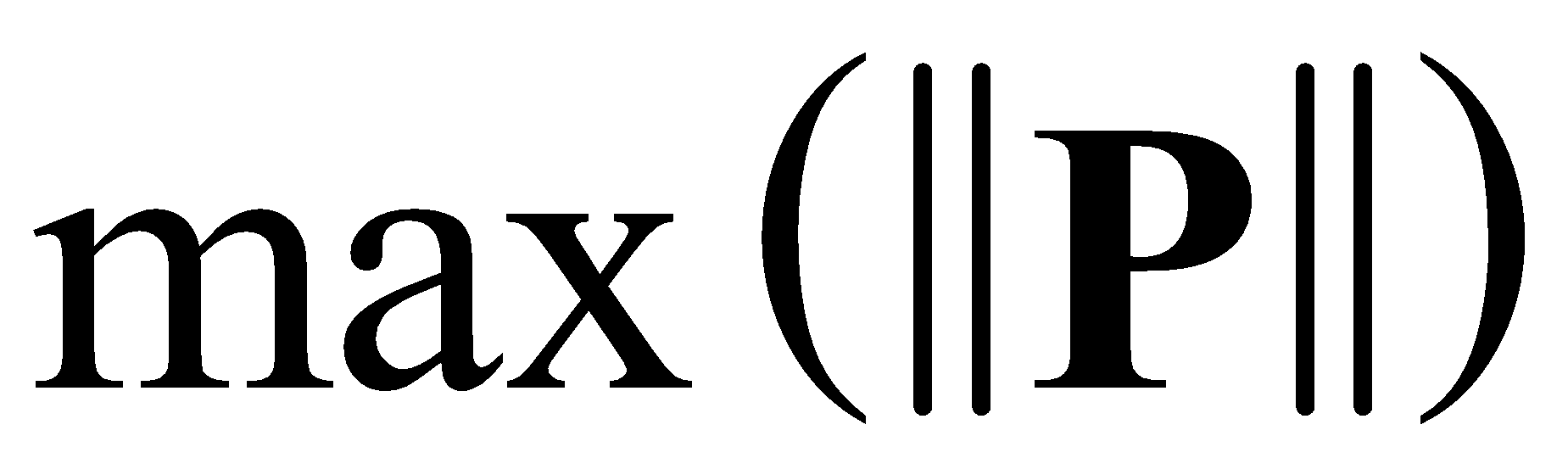
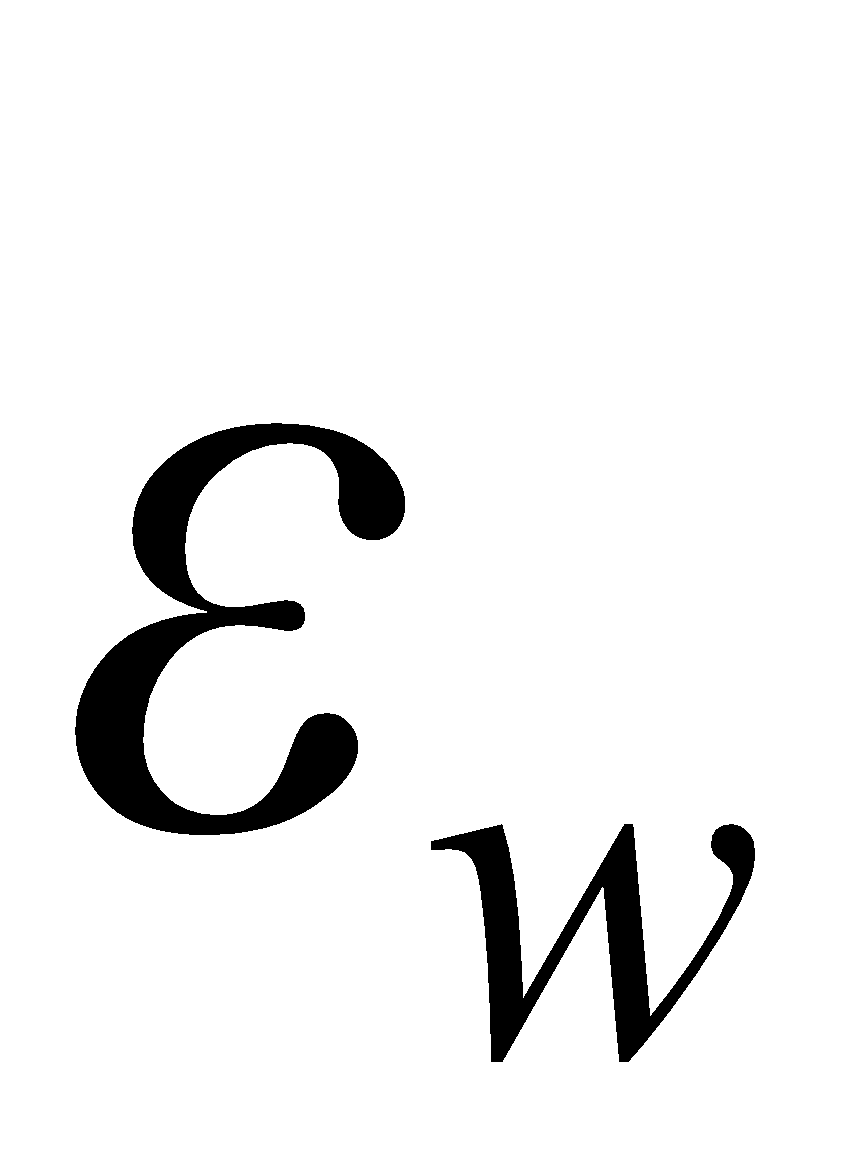




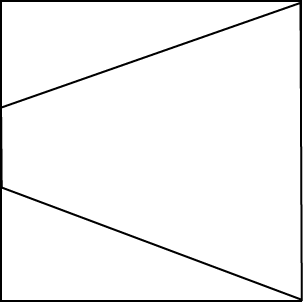
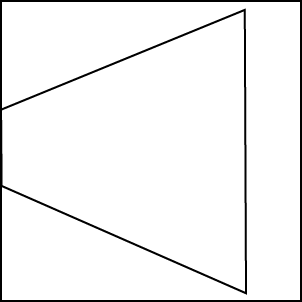
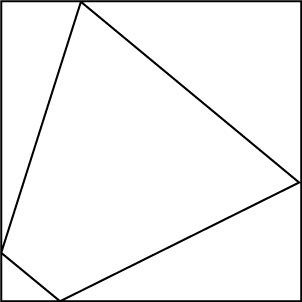




 (19.9)

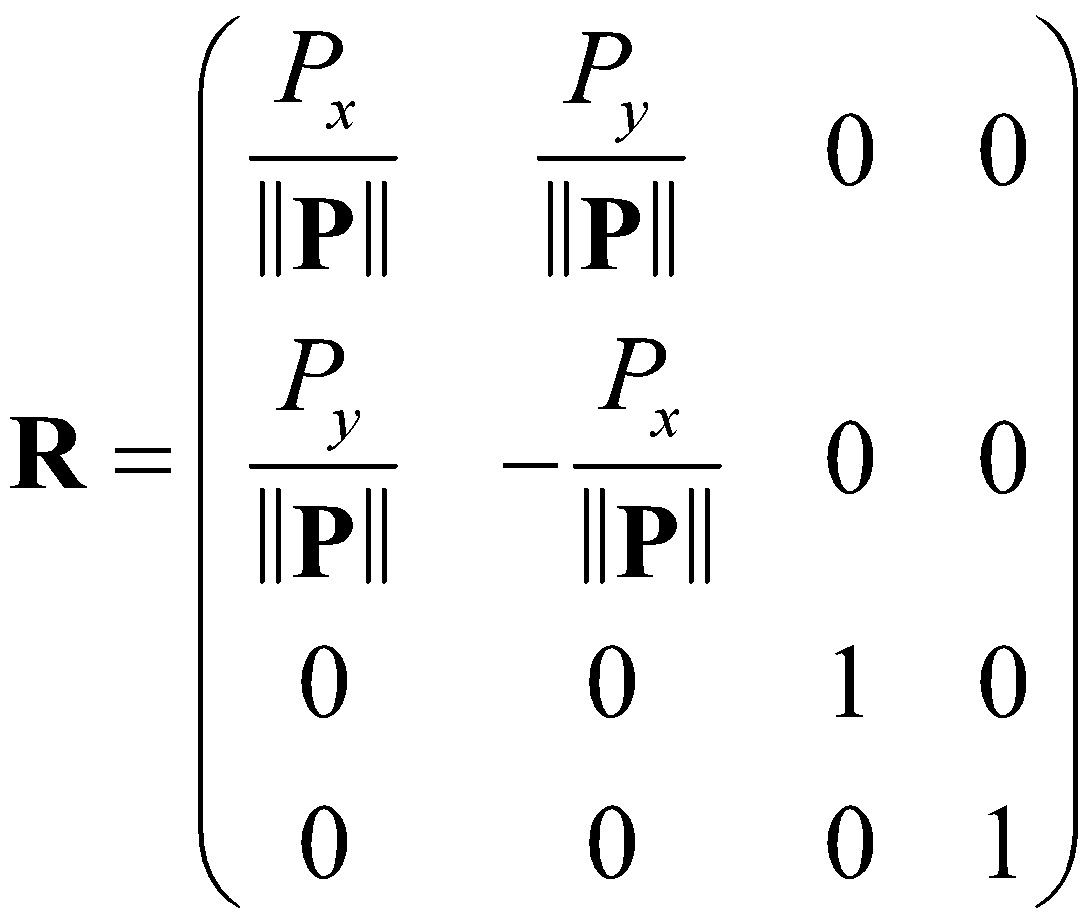
Clip  by  in order to control the back projection singularity. Modifying the free parameter  controls the near and far quality tradeoff for this worst-case scenario. This length of **P** effectively controls the position of the parameterization plane.

**Optimal shadow map orientation**

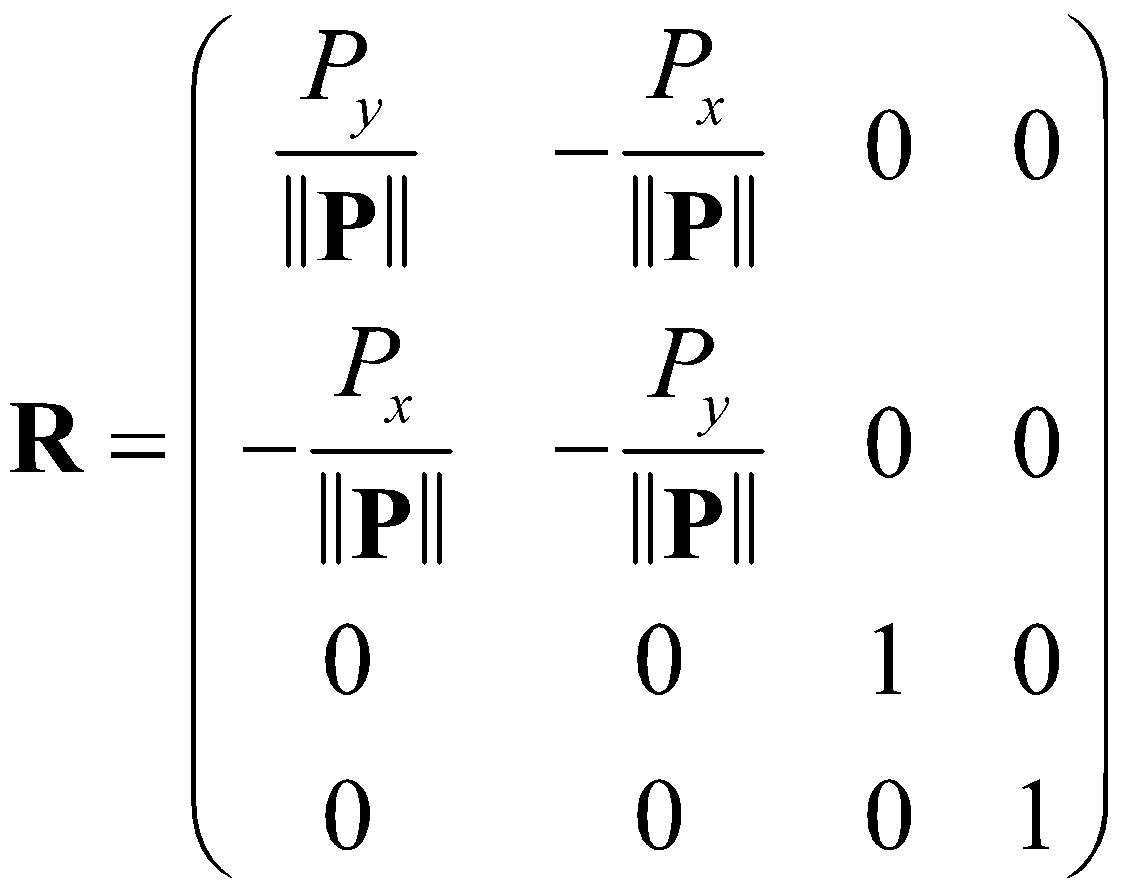


**Figure 19.2.** In the left diagram the frustum is allowed to rotate arbitrarily. The middle diagram shows the left diagram’s frustum rotated to be aligned with the *x*-axis, showing the shadow map waste. The right diagram shows that after applying the rotation matrix, the shadow map waste is reduced.

The projection vector in the shadow map can be pointing in any direction. This can lead to less than optimal shadow map area utilization. In order to get optimal shadow map area utilization, we can rotate the shadows such that the parameterization vector is always pointing down the *x*-axis. For an *x*-axis orientation of the projection vector:

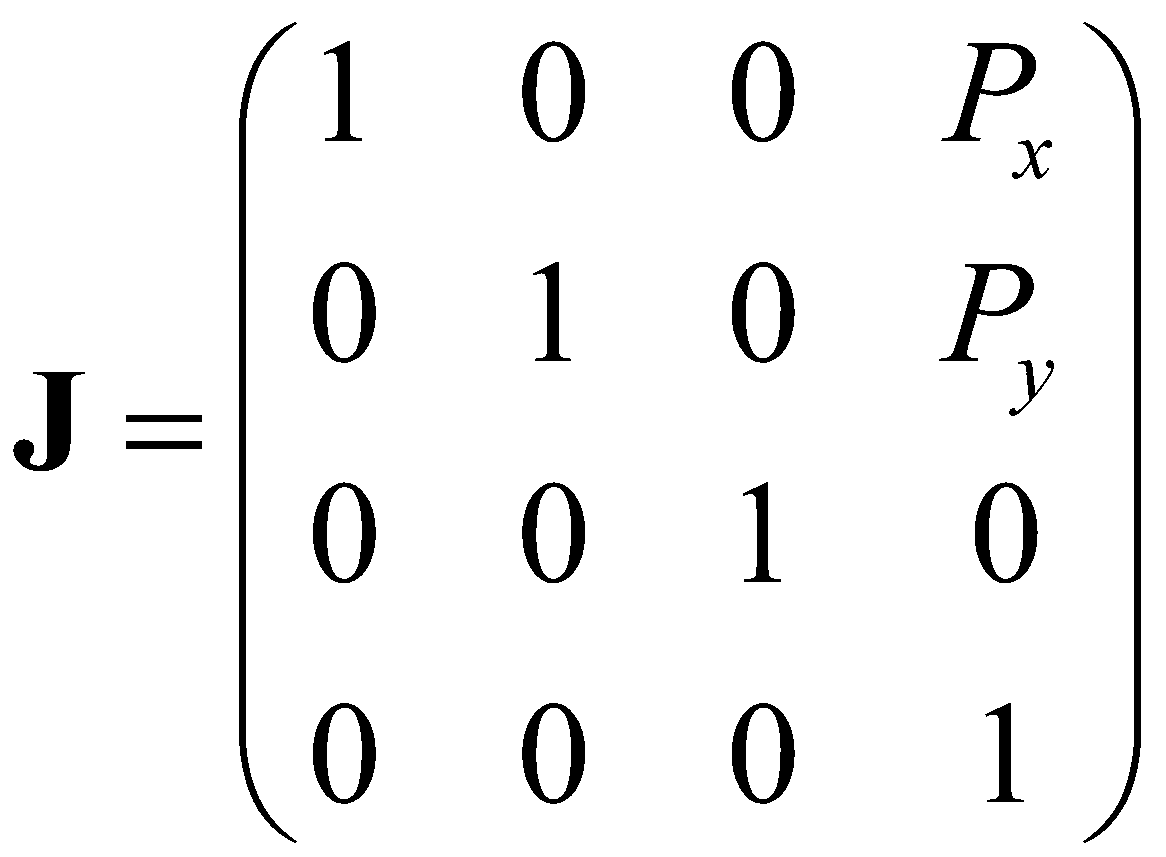
 (19.10)

If your shadow map is not of a box size and the *y*-axis has more resolution than the *x*-axis, then you may want to orient your shadows along the *y*-axis instead. For a *y*-axis orientation of the projection vector:

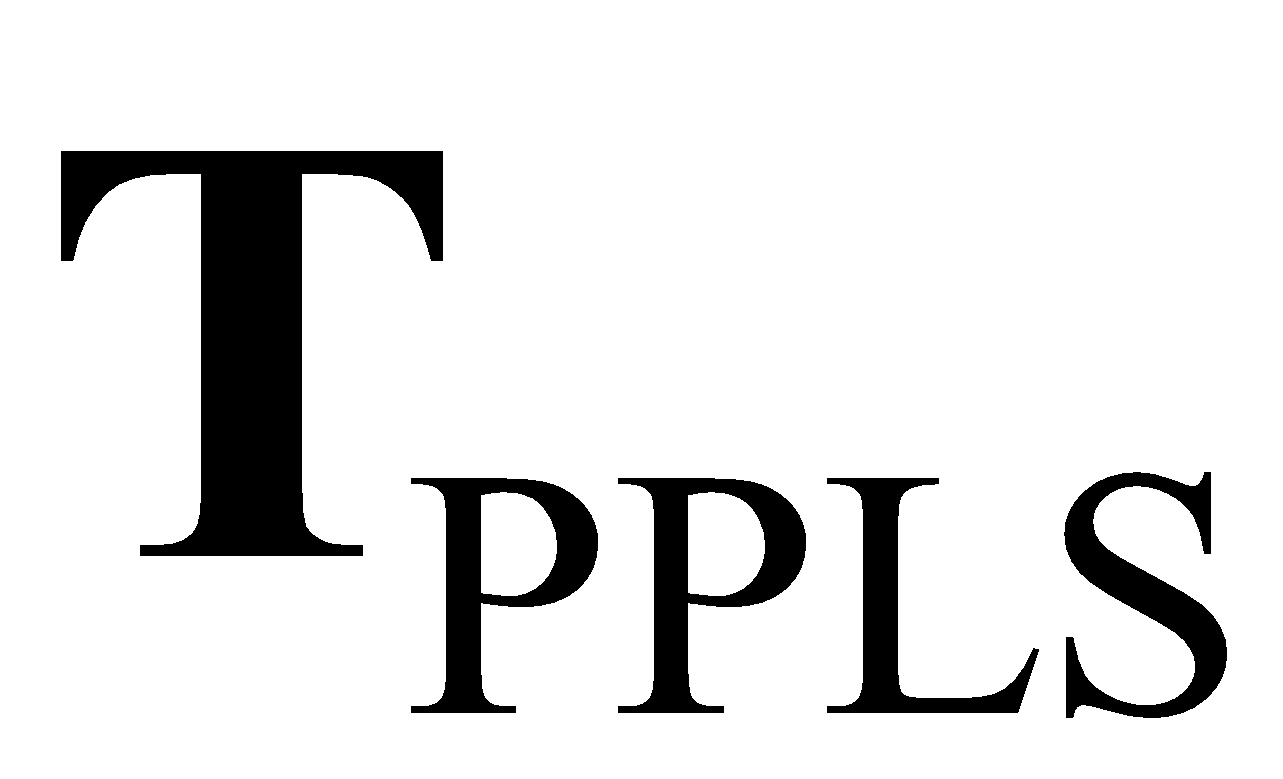
 (19.11)

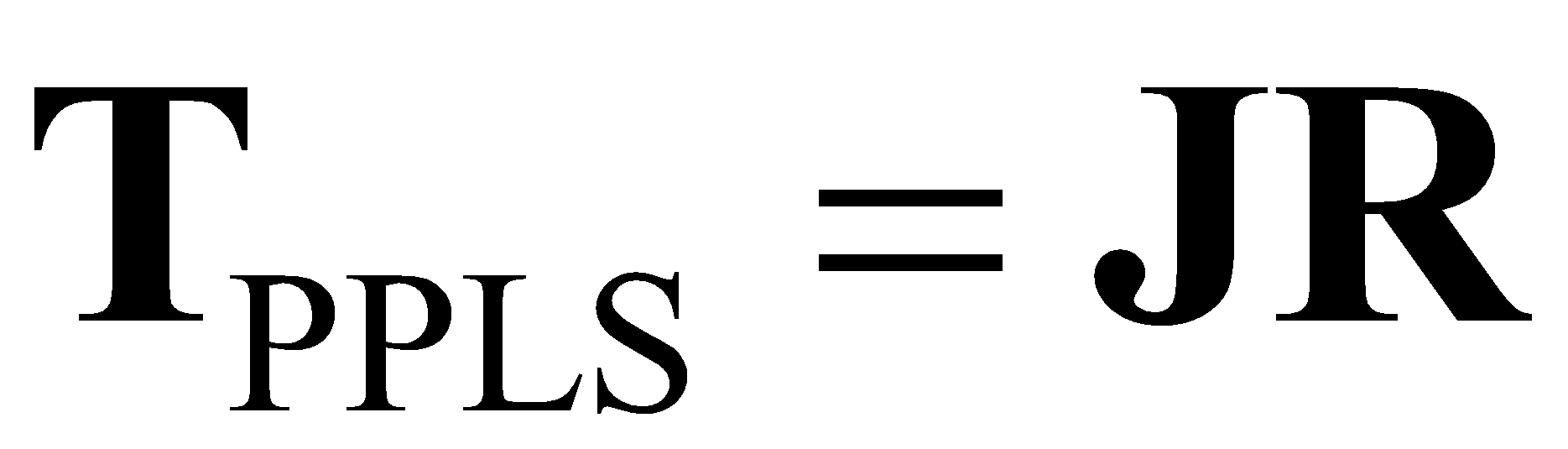
**Projection matrix**

The matrix that performs the warping of the shadow map space is as such:

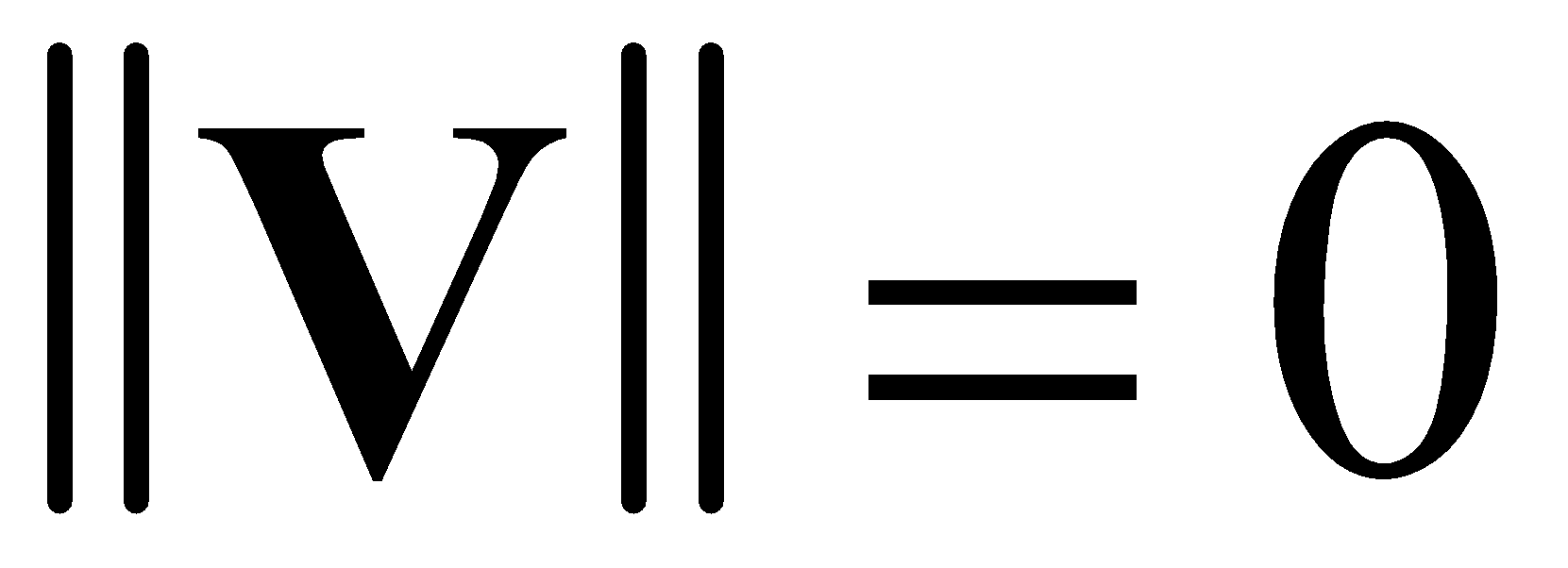
. (19.12)

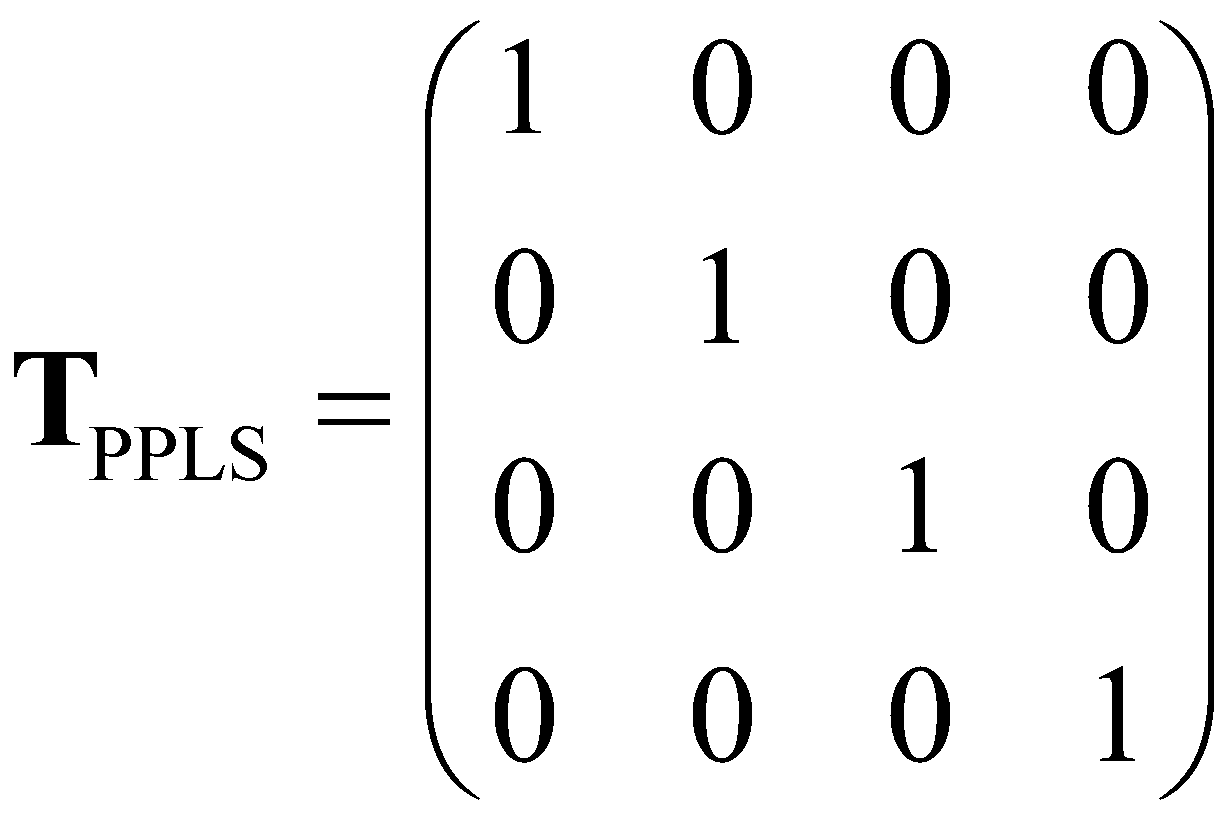
The shadow map warp happens in the divide by *w* that is usually performed in any standard GPU pipeline after the vertex pipeline but before the pixel pipeline.

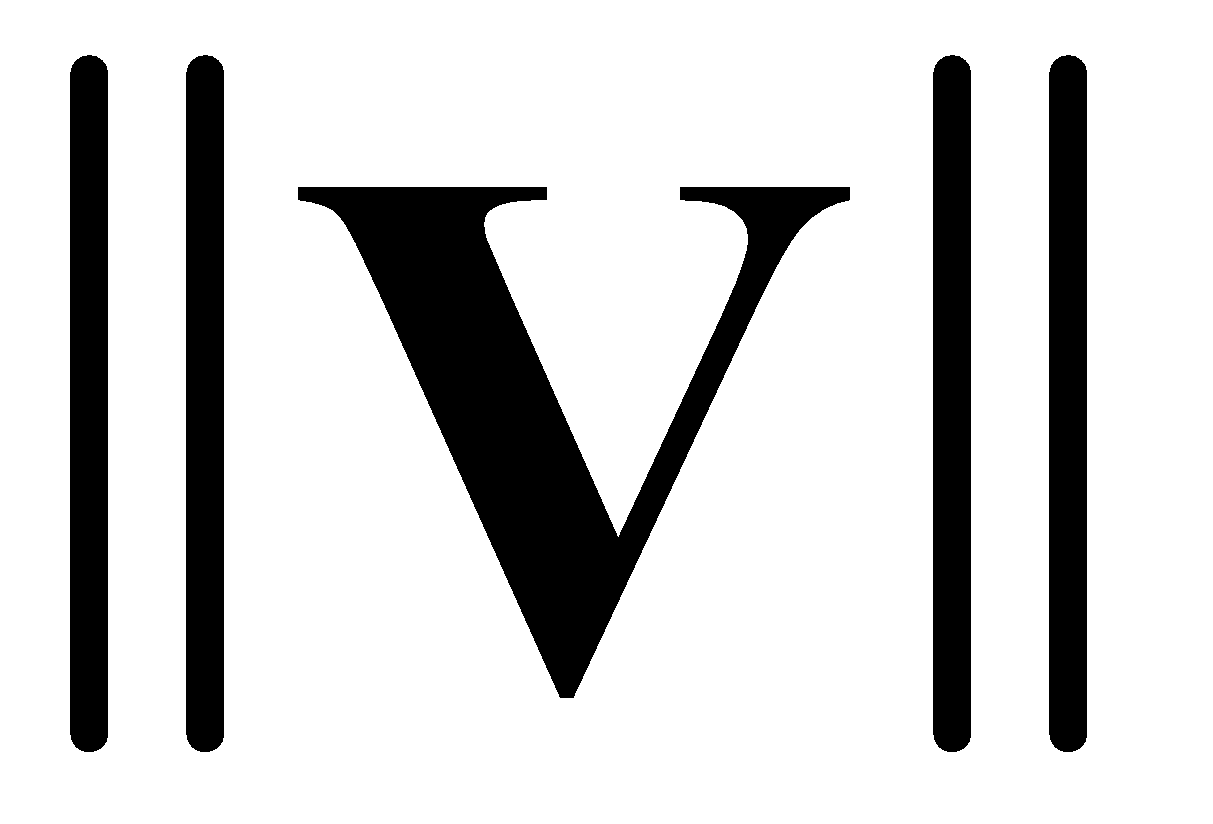
The transform  to post-projection light space is then

. (19.13)

**Miner’s lamp**

When the view direction is parallel to the light direction,  and mathematical exceptions occur. In this case, the typical solution is to fall back to uniform shadow maps as the solution degenerates to this case anyway. This can be achieved by setting

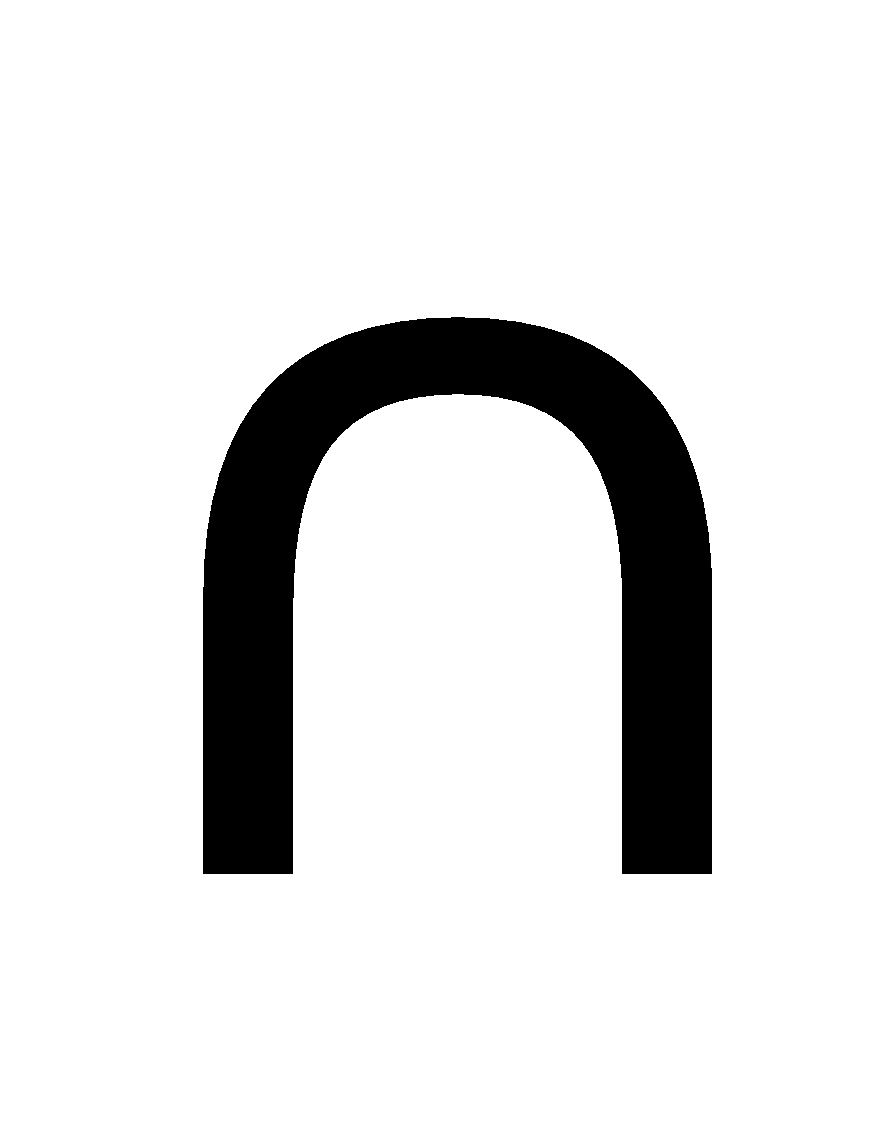
 (19.14)

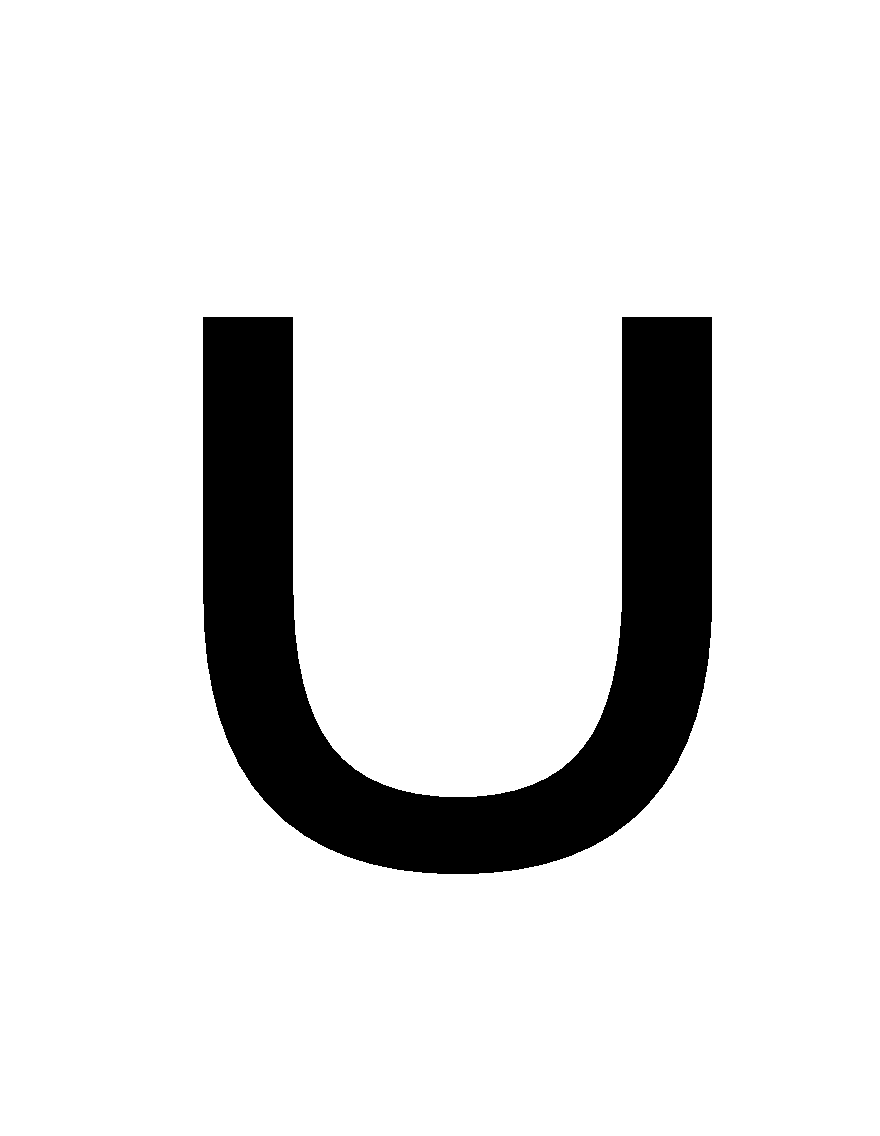
when  is close to 0.

**Unit cube transform**

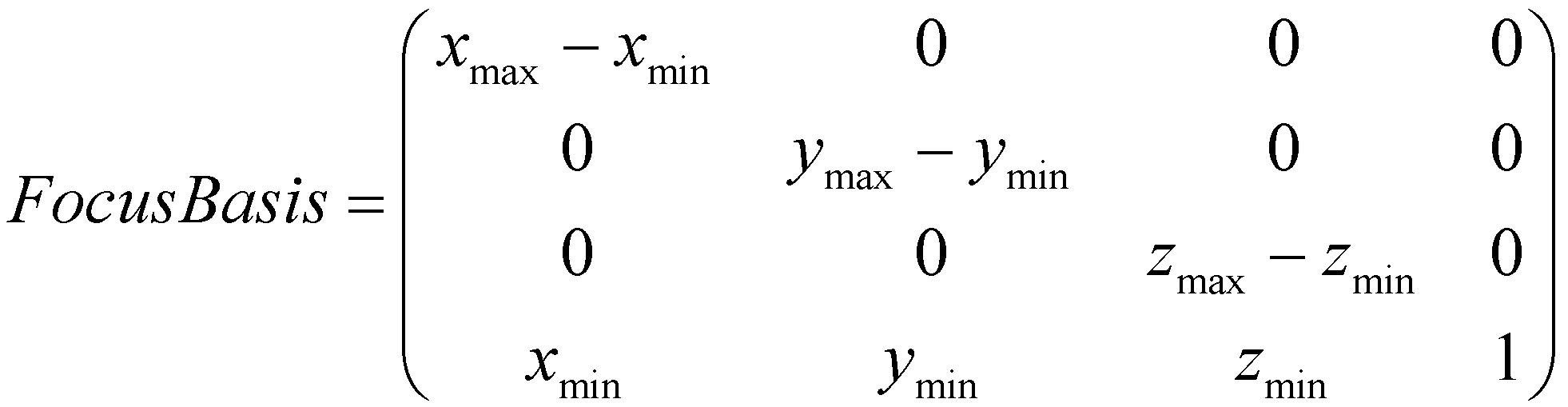
The focus region is the bounds of light rays where they intersect at least one caster and receiver.

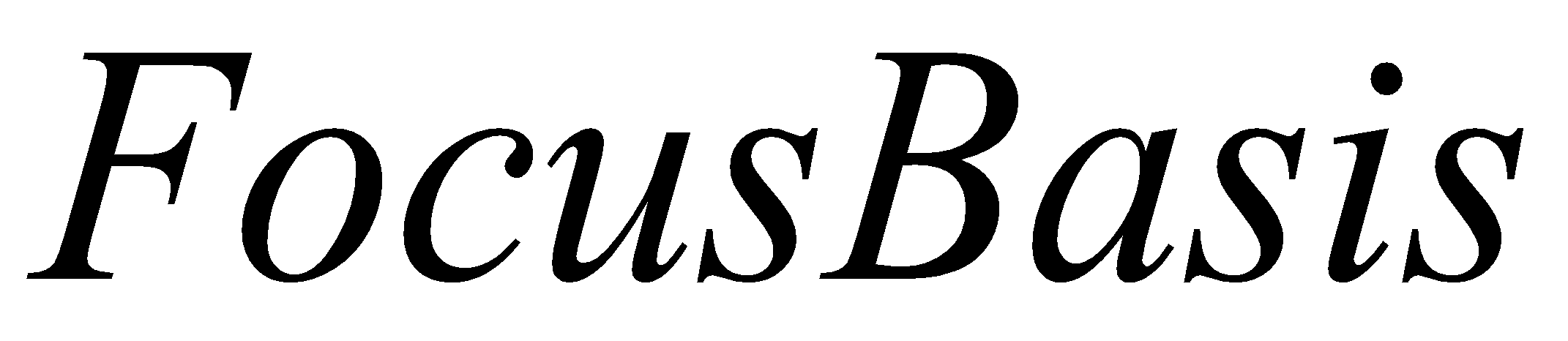
Lets define *Receivers* as the set of points of the bounding volumes of all shadow receivers transformed into PPLS space. Lets also define *Castors* as the set of points of the bounding volumes of all shadow castors transformed into PPLS space. The *x*- and *y*-axis of the focus region AABB is the intersection in the *x*-*y* plane of the *Receivers* AABB and the *Castors* AABB. The *z*-axis of the focus region AABB is the union of the *Receivers* AABB and the *Castors* AABB.

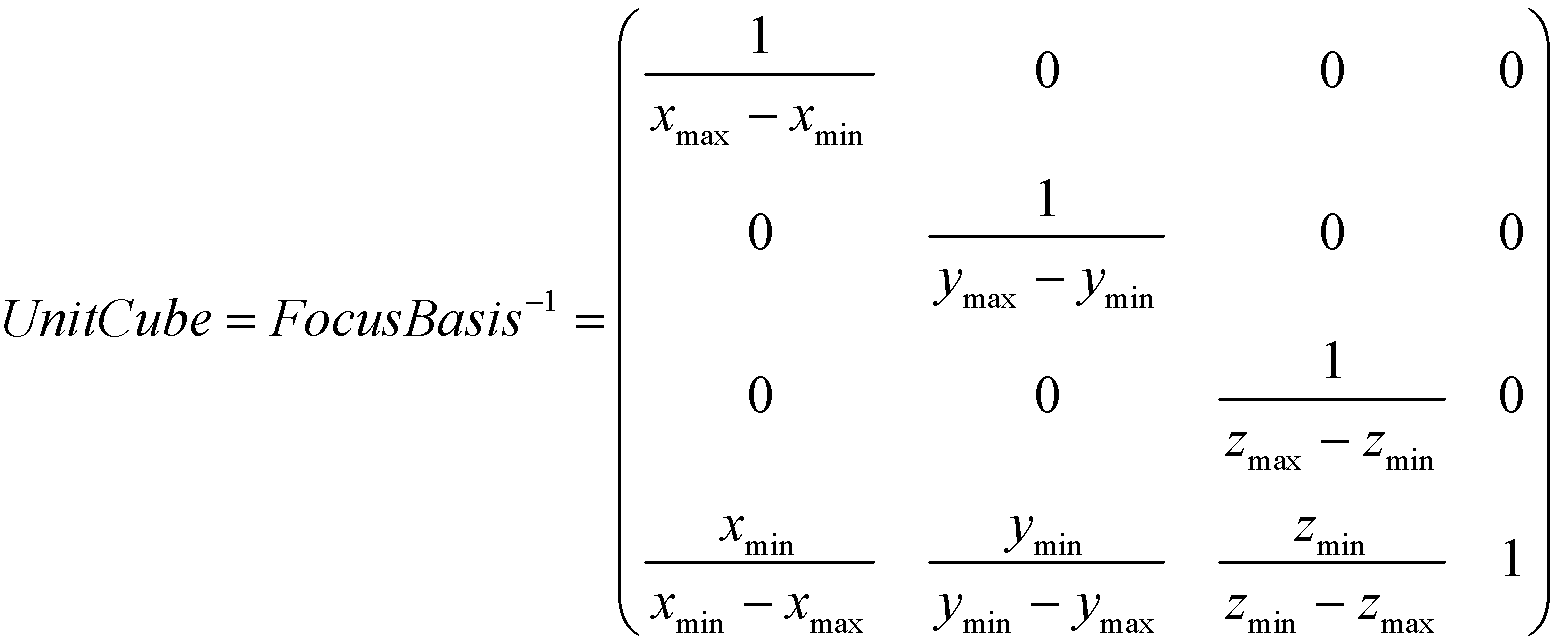
FocusAABBxy = ReceiversAABBxy  CastersAABBxy

FocusAABBz = ReceiversAABBz  CastersAABBz

Next, construct a linear basis that does not affect the warping property with the FocusAABB.

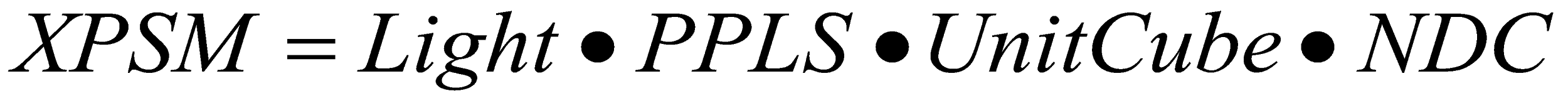
 (19.15)

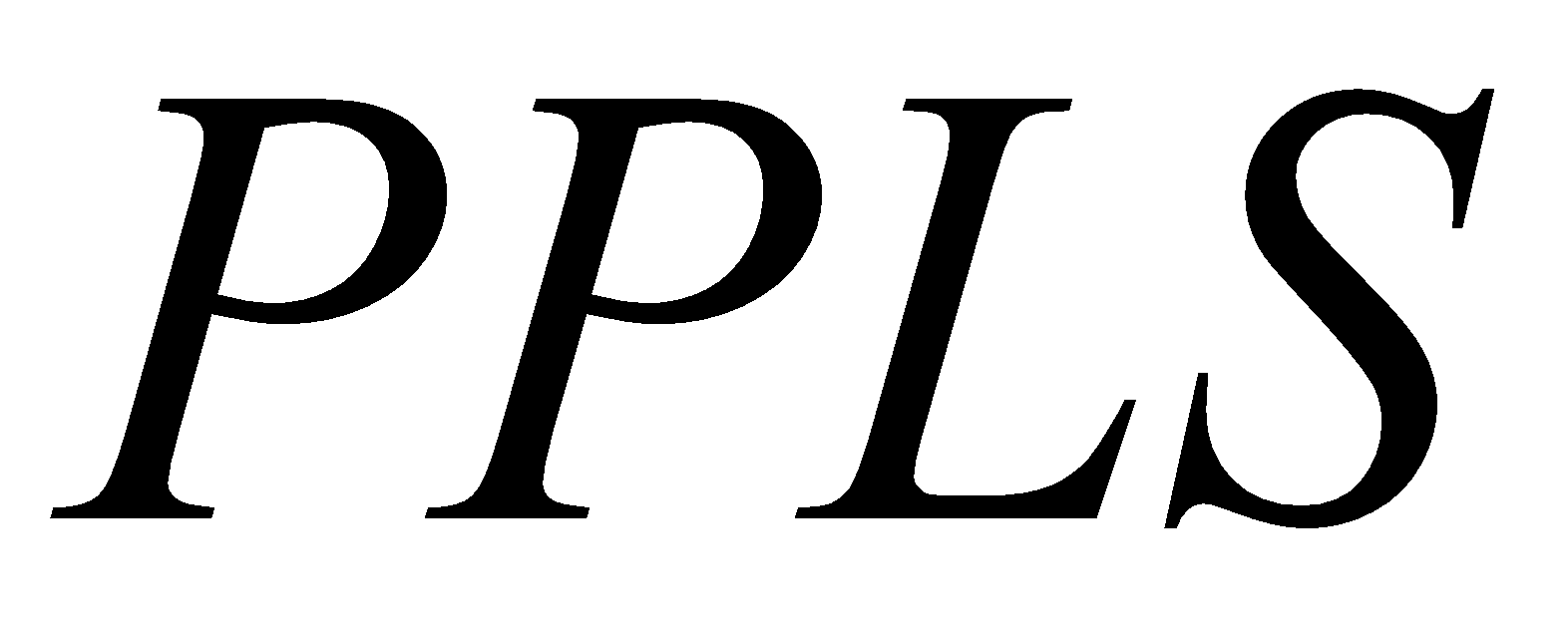
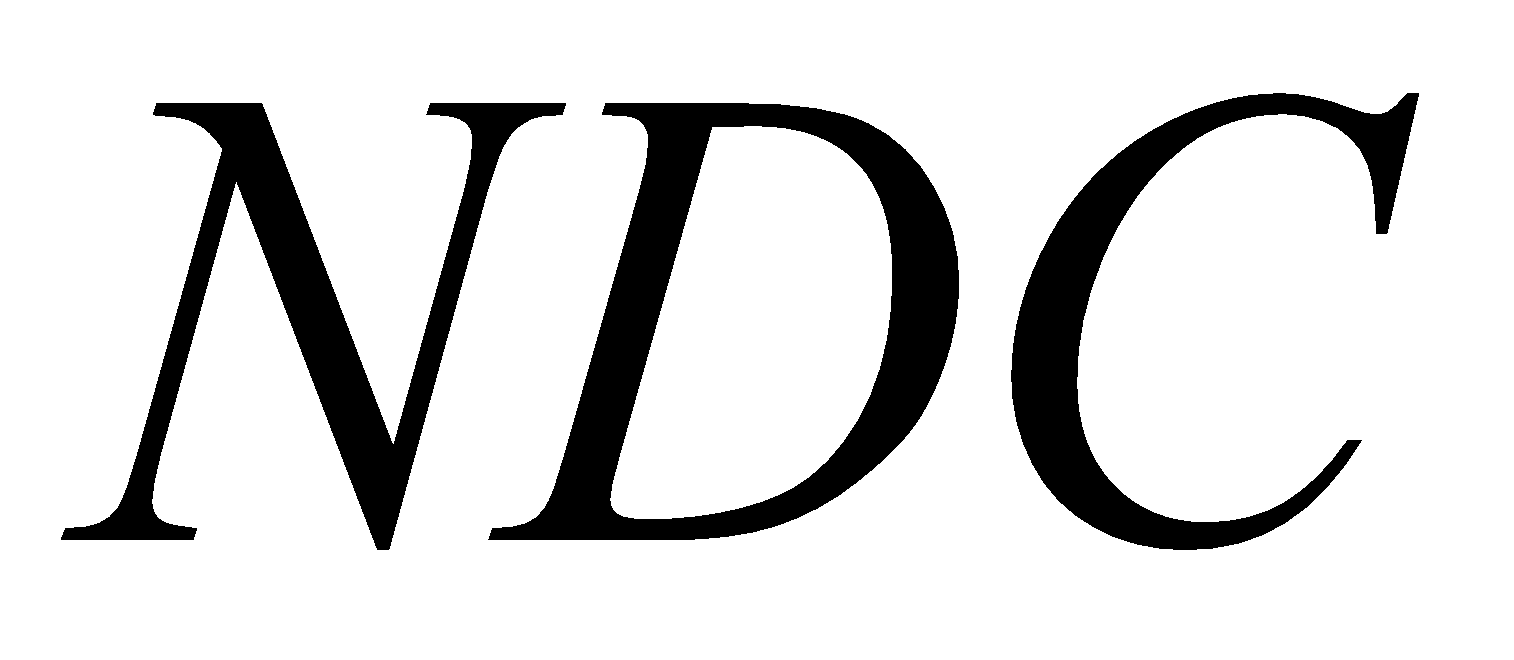
The unit cube transformation would then be the inverse of .

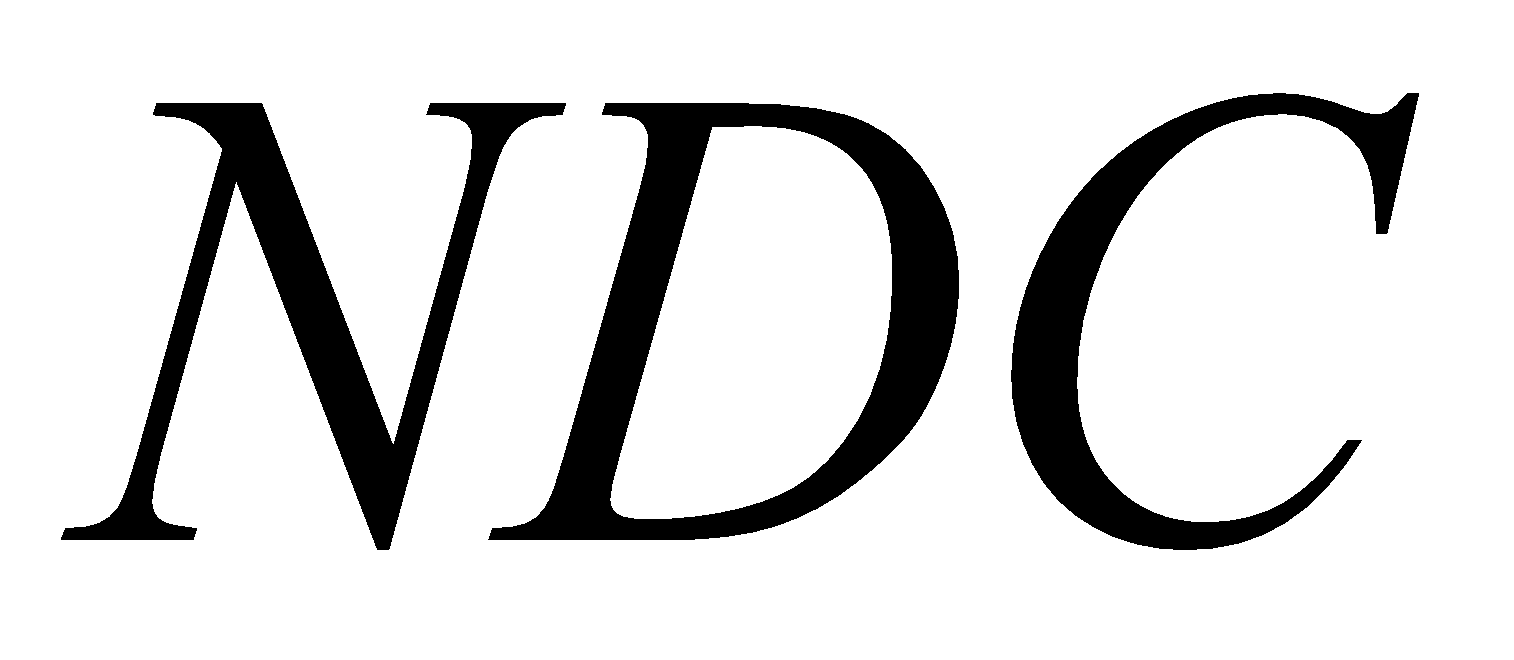
 (19.16)

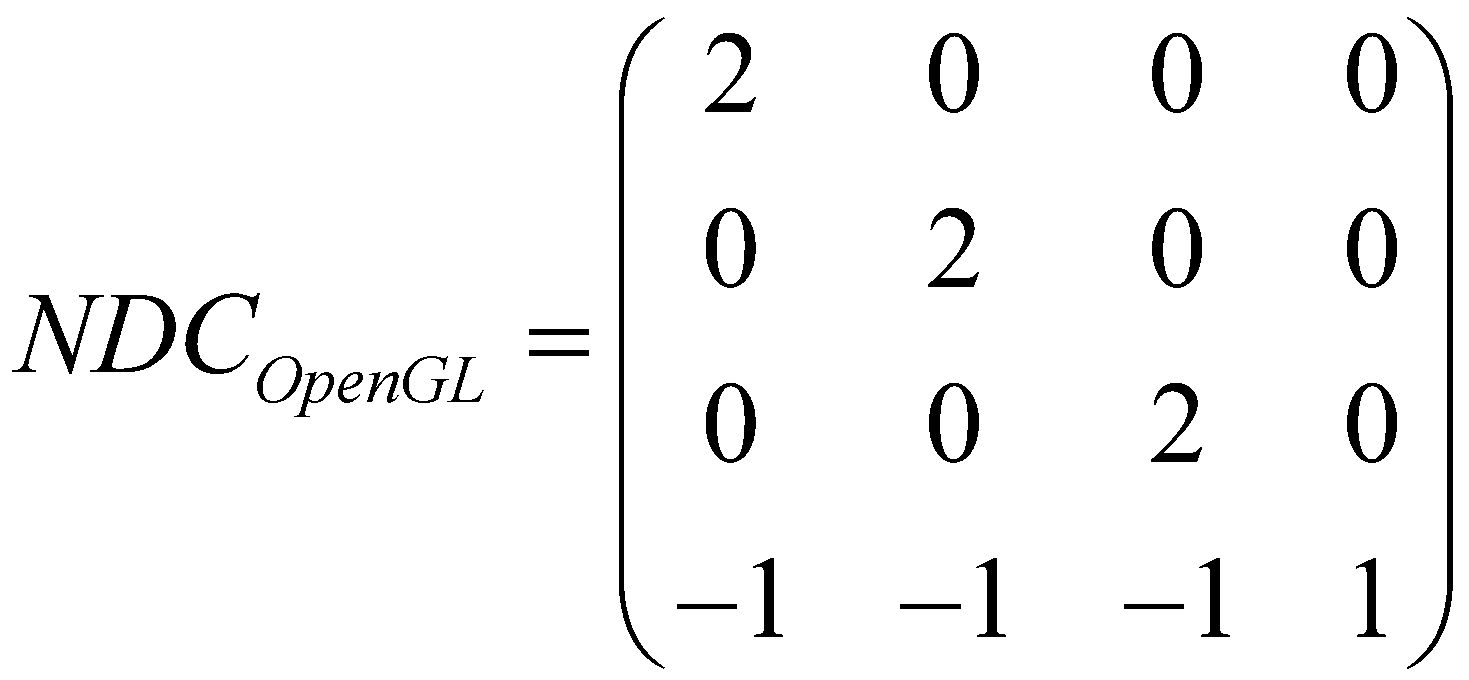
**Final transformation**

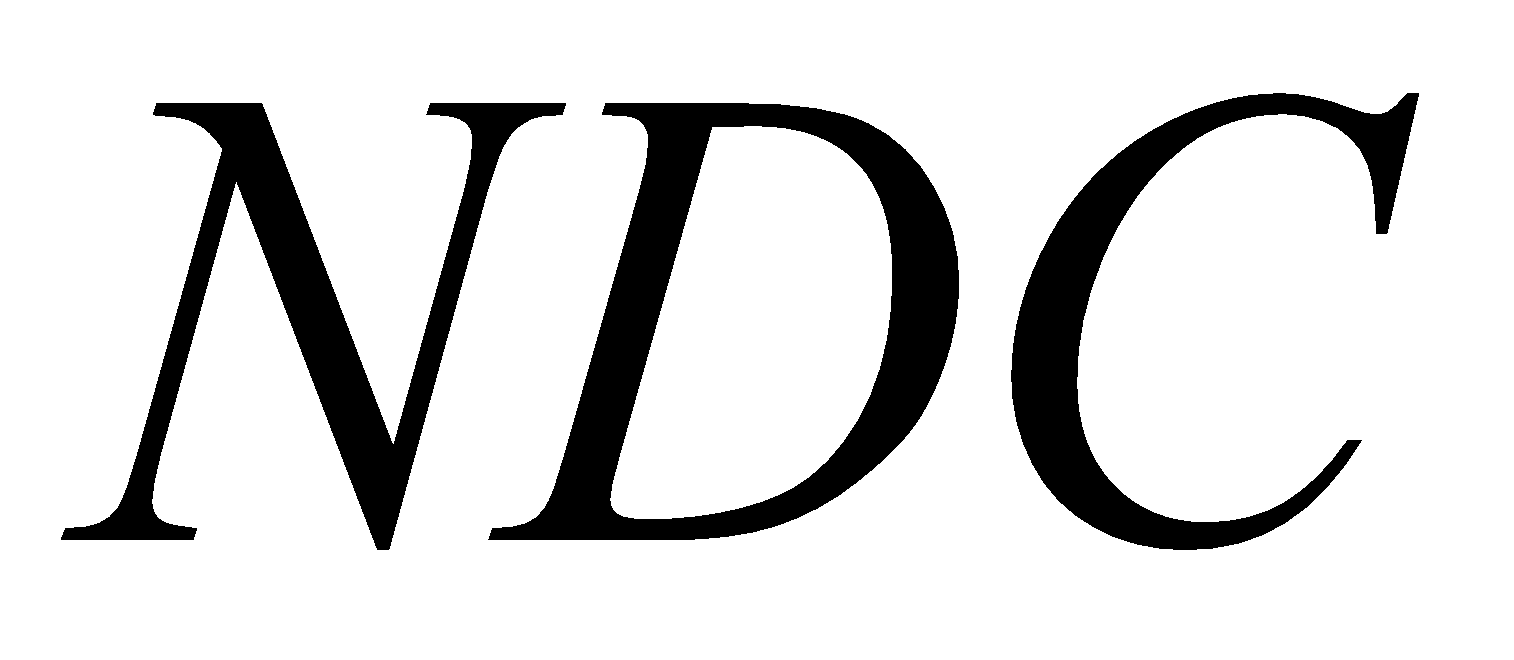
The final XPSM transform is:

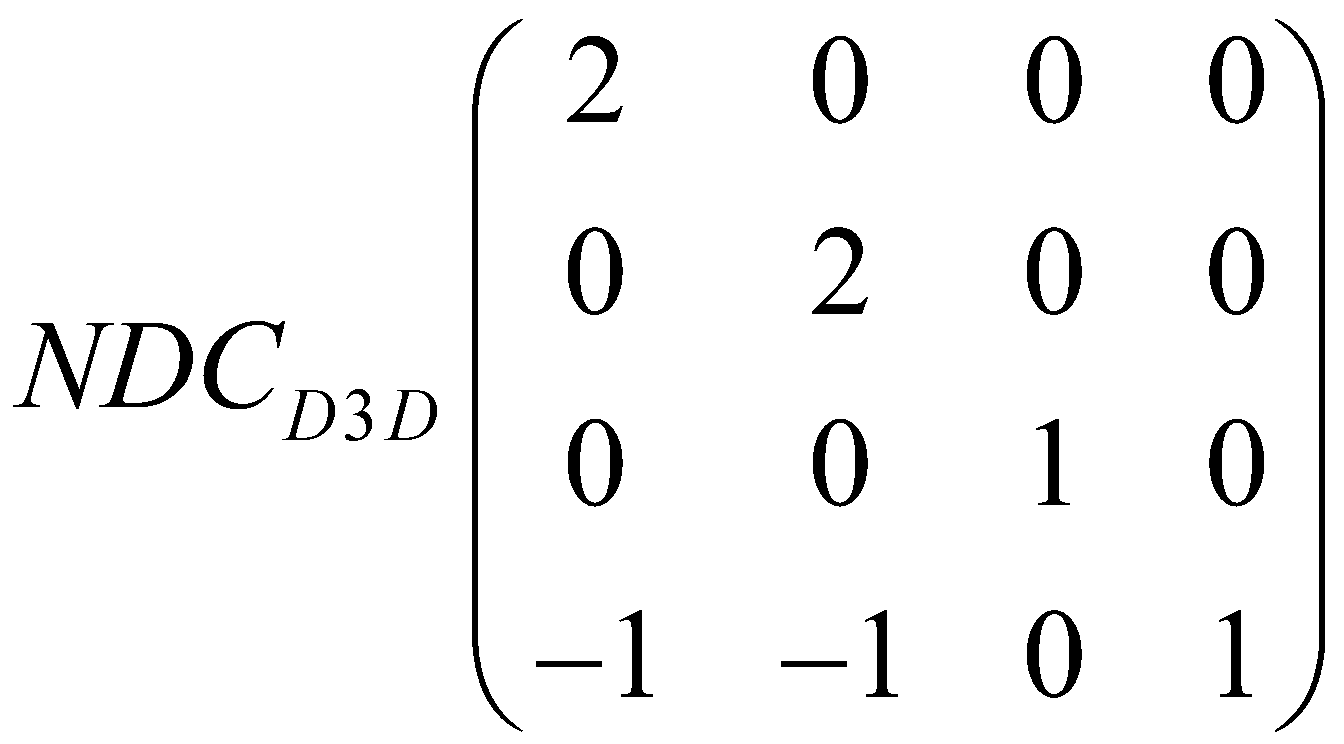


Where *Light* is the transformation into light space,  is the post projection light space transform, *UnitCube* is the transform into a unit cube, and  is a transform into OpenGL/Direct3D Normalized Device Coordinates.

The  transform for OpenGL is:

 (19.17)

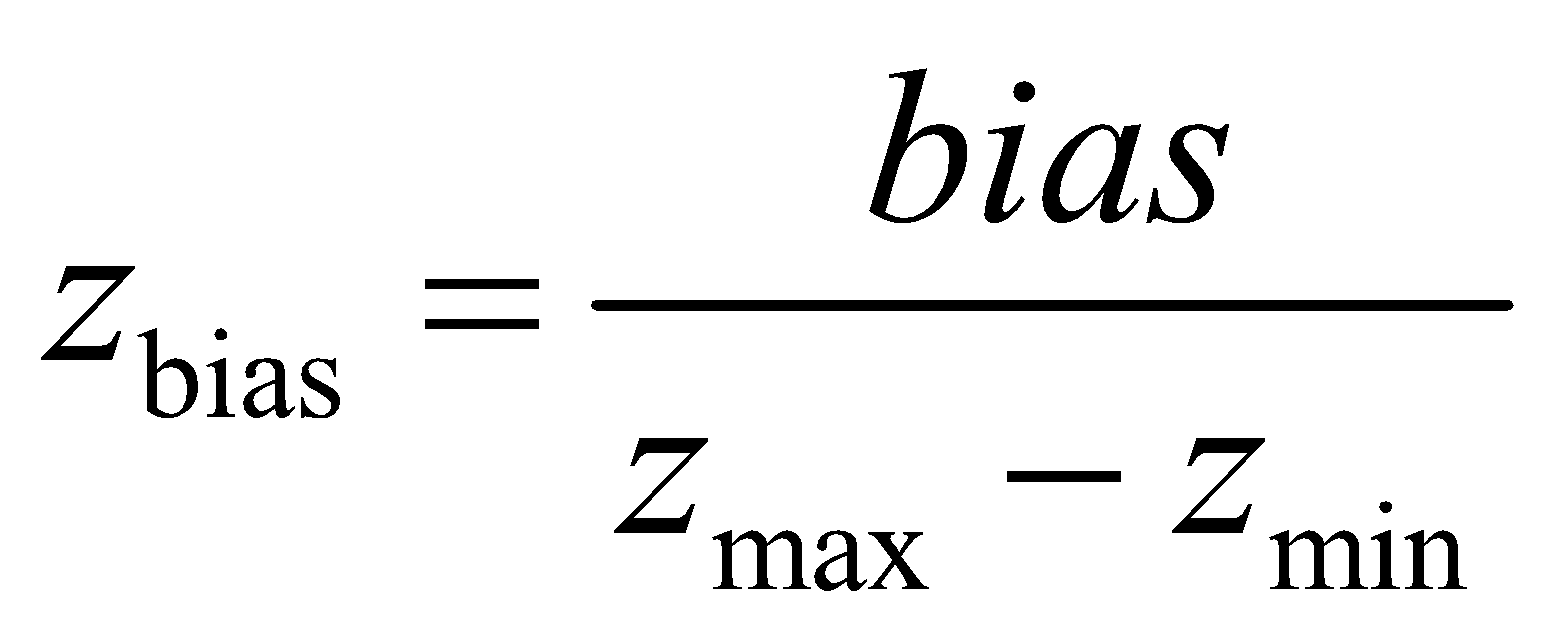
The  transform for Direct3D is:

 (19.18)

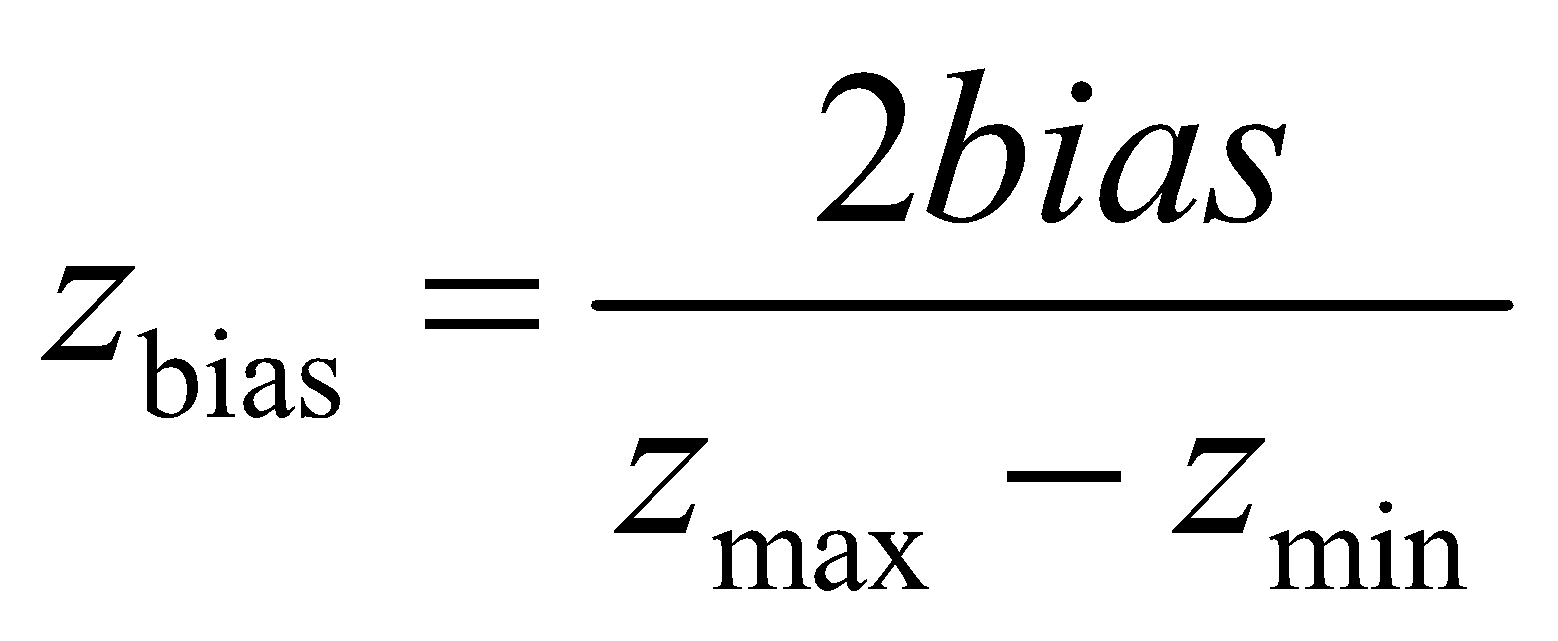
***Z* bias**

The problems with using *z* bias are not uncommon to perspective shadow maps. This is due to the fact that the divide by *w* causes the shadow map *z* values to be nonlinear in the *x*-*y* plane. The typical solution involves biasing before the divide by *w* where the *z* values are linear. XPSM is no exception and biasing should be done just before the perspective divide.

For Direct3D, *z* bias is computed as

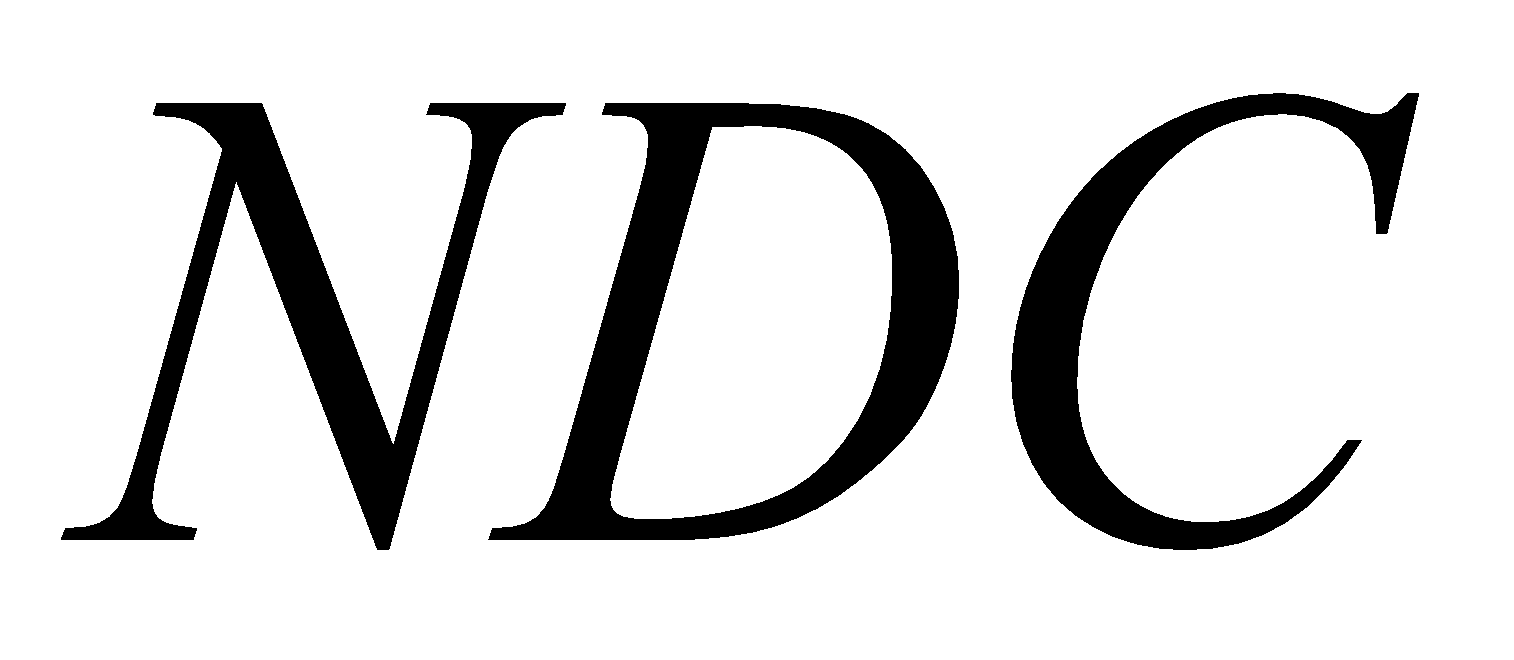
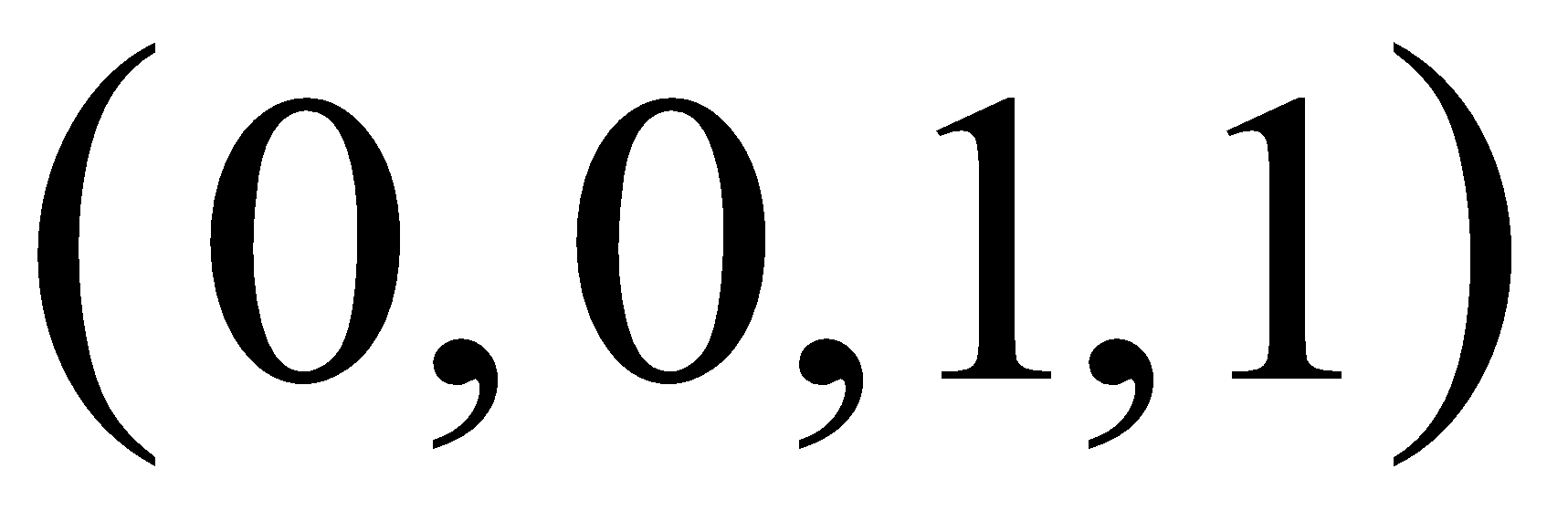
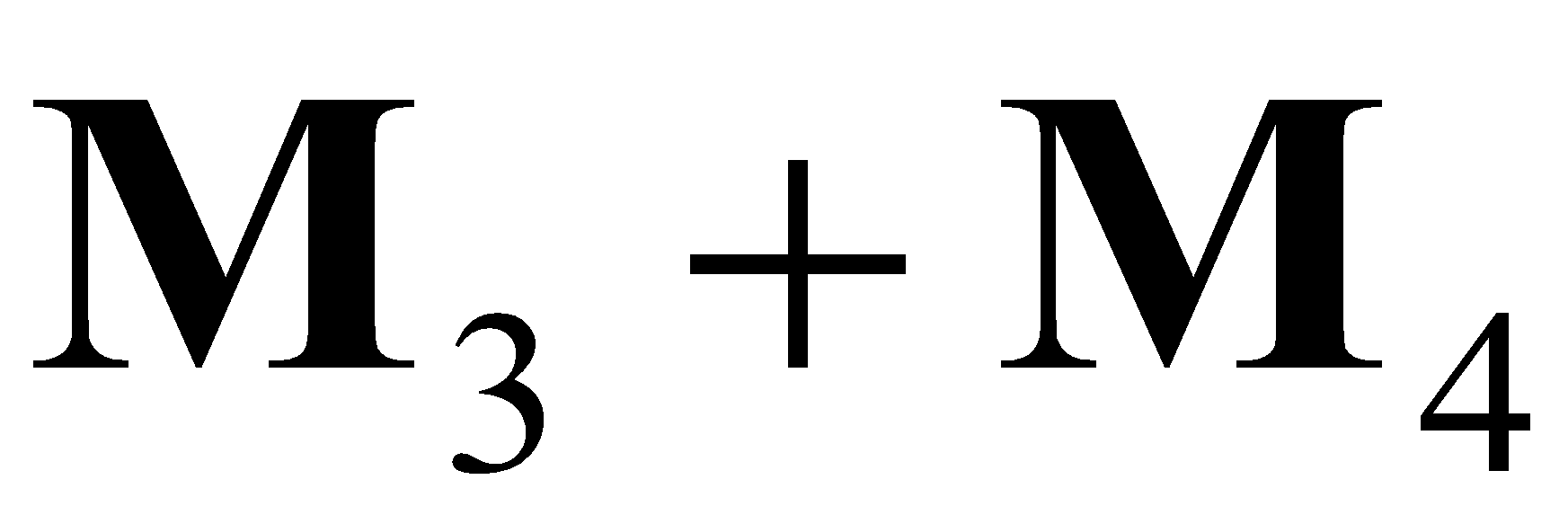
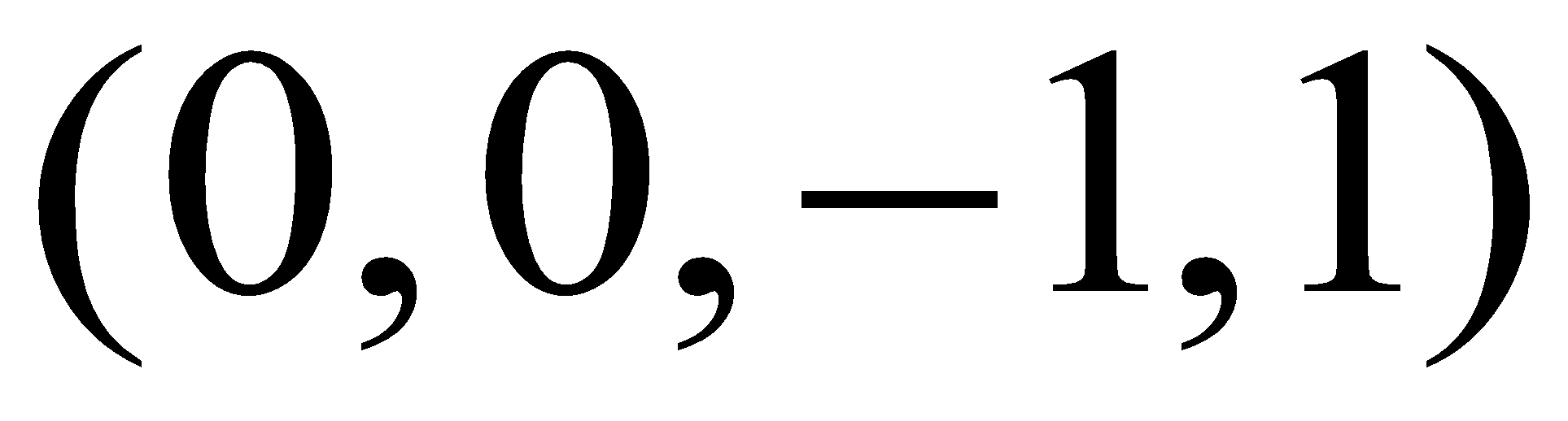
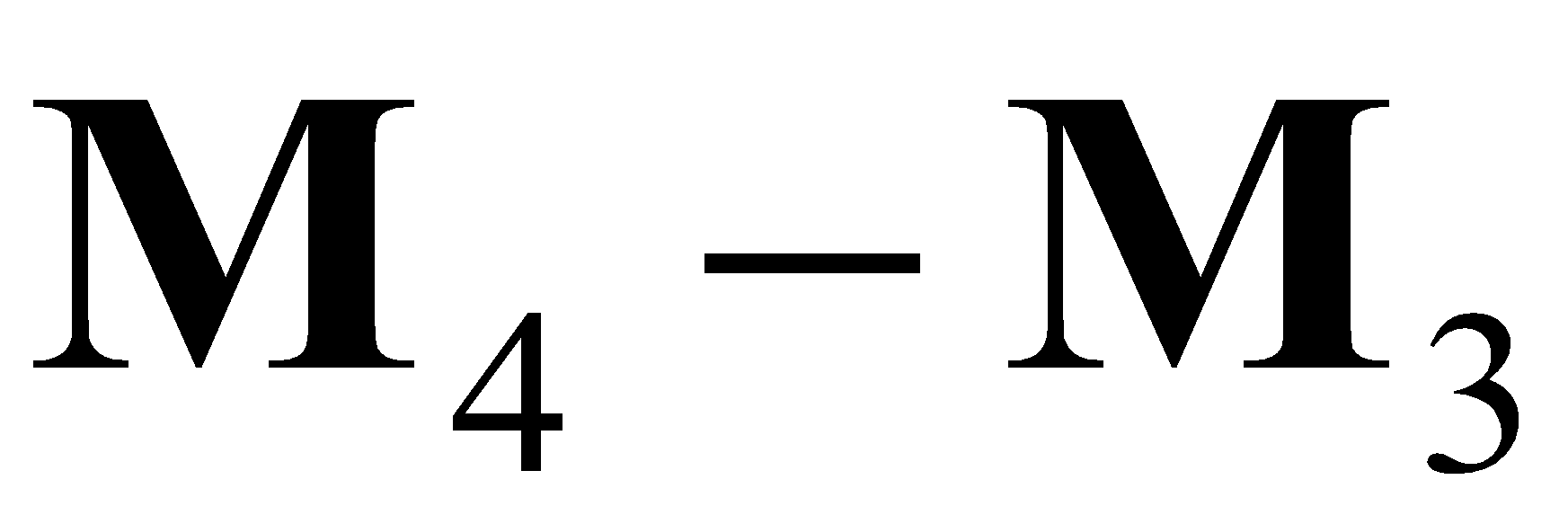
 (19.19)

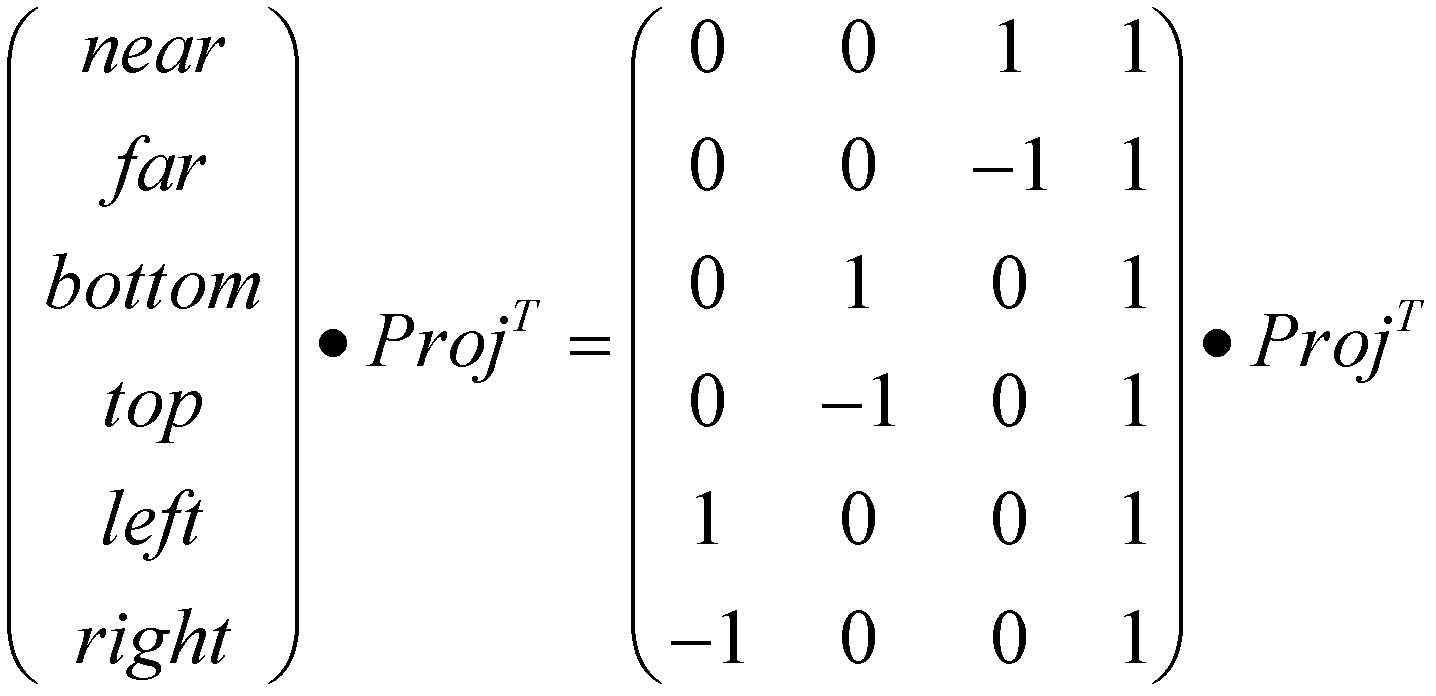
For OpenGL, *z* bias is computed as

 (19.20)

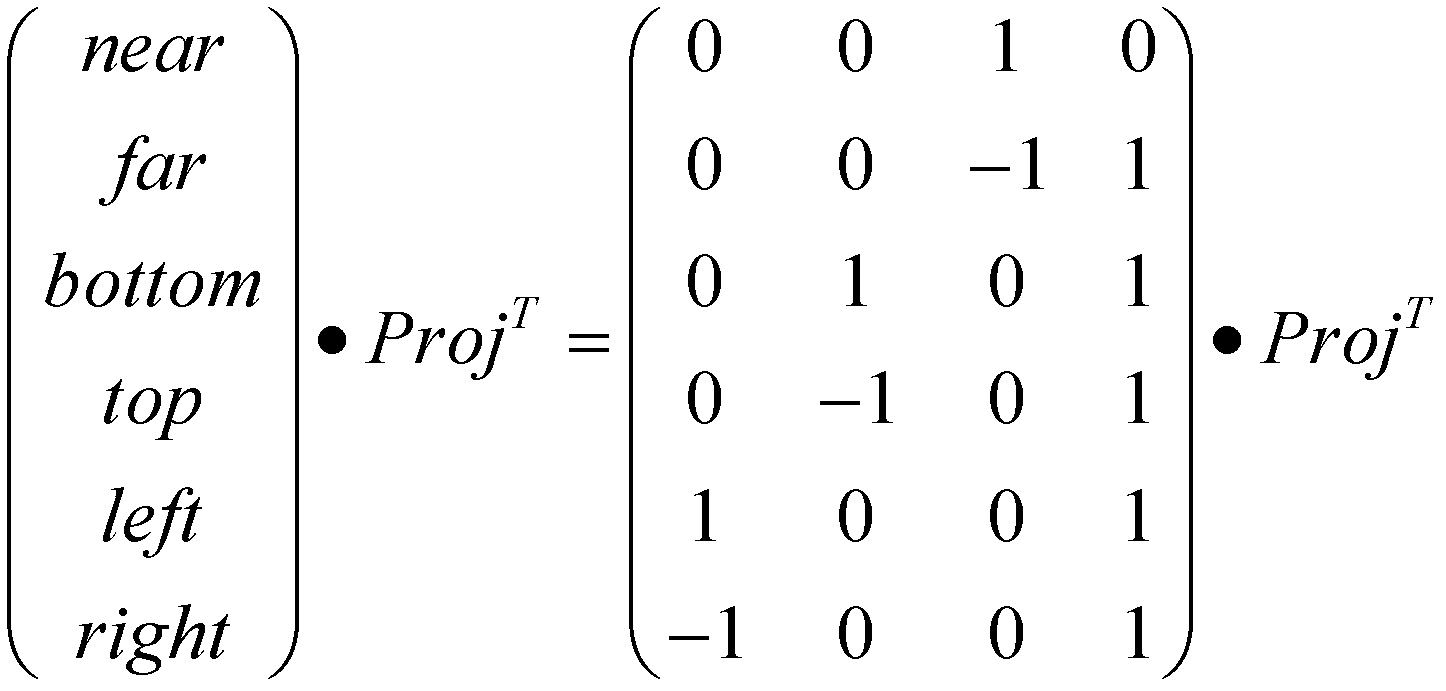
The times two for OpenGL is due to different normalized device coordinates.

**Frustum plane extraction**

Extracting frustum planes is important to culling as well as visualization. Planes are transformed from  to view space by the transpose of the projection matrix. In OpenGL, the near plane equation is , and so the plane in camera space is , the far plane equation is , which in camera space is , etc. This can be expressed concisely in matrix form for OpenGL

 (19.21)

and for Direct3D

 (19.22)

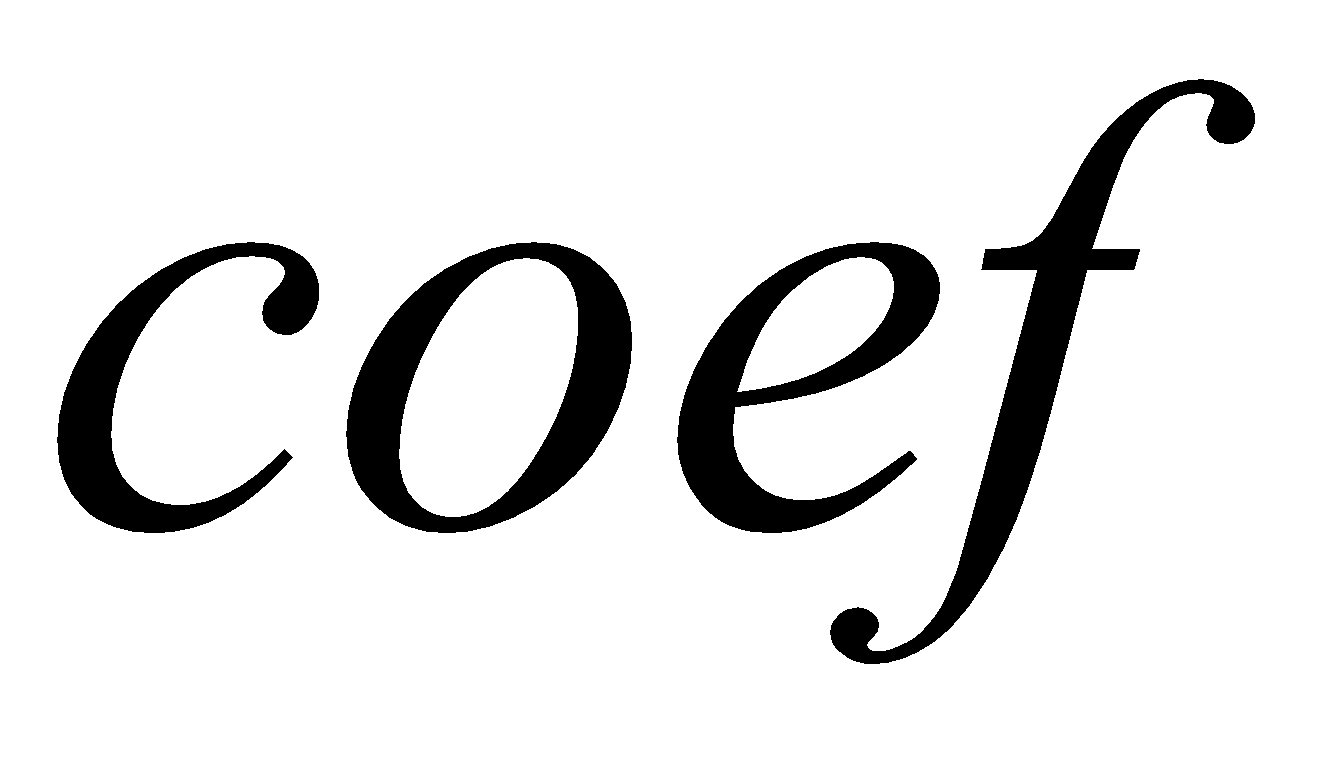
The result of the matrix multiplication is a matrix of plane equations representing the various frustum planes in view space.

**19.3 Results**

XPSM has been implemented in Nvidia’s Practical PSM demo (can be downloaded from http://xmvlad.110mb.com/) by Gussev [7], the original inventor of the technique. On average, XPSM has better quality than PSM and LiSPSM with shadow resolutions comparable to TSM with the proper parameters. In worst-case scenarios XPSM has better quality than PSM/LiSPSM and doesn’t produce the typical PSM/TSM visual artifacts (shadow acne, etc.) or singularities, with the exception of unstable shadow swimming which is typically solved with filtering or higher resolution shadow maps. PSM/LiSPSM/TSM/XPSM all suffer from this swimming artifact. The lack of singularities is primarily due to the tunable nature of controlling the negative effects. In practice, tuning these magic numbers is a fairly easy process and so far has not caused any headaches. XPSM is a good practical drop-in solution for directional lights with shadow maps that consistently out-perform typical uniform shadow mapping solutions and is patent free.

**19.4 Future Work**

Through the property that successive frames are very similar to each other, analyzing a downsampled depth buffer should enable the optimization of the parameterization vector as to asymmetrically distribute the shadow resolution when it is advantageous to do so. This is an avenue of research for XPSM that has not yet been explored for improving shadow quality.

It should also be possible to dynamically change the  parameter as to improve shadow quality.

Splitting up the view frustum into view parallel slices can better approximate the logarithmic nature of the depth buffer and provide enhanced quality. Implementation of this is left as an exercise to the reader, but is highly recommended.

**References**

[1] Williams L. 1978. Casting curved shadows on curved surfaces. In Proceedings of SIGGRAPH 1978.

[2] Martin T., Tan T.-S. 2004. Anti-aliasing and continuity with trapezoidal shadow maps. In Proceedings of Eurographics Symposium on Rendering 2004.

[3] Wimmer M., Scherzer D., Purgathofer W. 2004. Light space perspective shadow maps. In the Eurographics Symposium on Rendering 2004.

[4] Stamminger M., Drettakis G. 2002. Perspective shadow maps. In SIGGRAPH 2002 Conference Proceedings

[5] Chong H., Gortler S. 2004. A lixel for every pixel. In the Eurographics Symposium on Rendering 2004.

[6] Kozlov S. 2004. Perspective shadow maps: care and feeding. GPU Gems, Addison-Wesley.

[7] Gussev V. 2007. Extended Perspective Shadow Maps (XPSM). http://xmvlad.110mb.com/xpsm.pdf