Extended Perspective Shadow Maps (XPSM)

http://xpsm.org
Vladislav Gusev, 10.08.2007, xmvlad@gmail.com

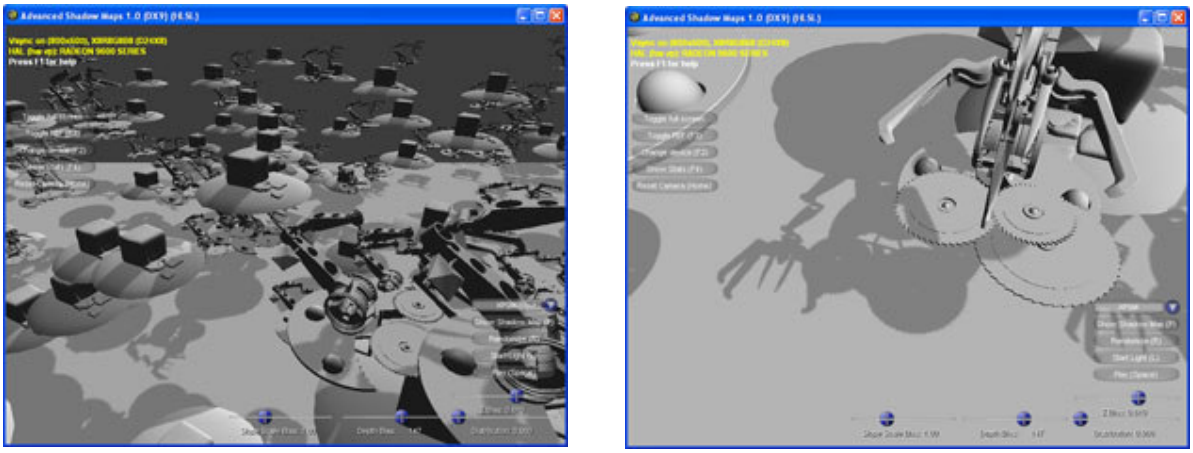


Figure 1: *XPSM results (~400 objects in a scene, 1536x1536 shadow map)*

1. Introduction

Shadows are one of the most important effects to convey realism in computer-generated scene. Shadow mapping [1] is one of the most popular and efficient approaches for real-time shadow rendering. The image-based nature makes shadow mapping extensively useful in real-time applications, despite they perspective aliasing problems. With shadow maps, we simply render the scene from the viewpoint of the light source, creating a depth image. The current depth buffer is stored as a shadow map texture. In the second pass, when rendering each pixel in the final image, the visible point is transformed into the light coordinate system. If the distance in shadow map is less than distance between the point and the lighting source, than pixel is shadowed.

One well-known drawback of shadow mapping is perspective aliasing which typically happens when the user zoom into a shadow boundary so that single shadow map pixels become visible. A number of papers have tried to solve perspective aliasing problem. The most prominent for real-time computer graphic of these is TSM [2], LiSPSM [3], PSM [4], and Plane optimal shadows [6]. But all of these methods suffer from some problems and don't provide practical drop-in solution for high quality shadows rendering: TSM ("+": very good quality, "-": z acne problem, high complexity, patented), LiSPSM ("+": good quality, medium complexity, "-": significant degradation in worst case), PSM ("+": good quality, "-": high complexity), Plane optimal shadows ("+\-": optimal quality, but only for very special case). What is more disturbing that the main source of artifacts - Z bias non-linearity in post perspective space even not addressed in practical manner in academic papers, with only notable exception going from Kozlov S. [5].

In this paper, qualitatively new approach to shadow map parameterization is presented. If you are not familiar with projective geometry look at brief introduction to this subject [7], [8] though it is not necessary, but will help to clearly understand essence of this method. In terms of projective geometry every transformation can be decomposed into a vector perpendicular to infinity hyper plane (in R^4 embedding of P^3) we name it – *projection vector* and some affine transformation. Thus we approach solution in two steps: at first, find optimal warping effect by fixing projection vector, at second, find some affine transformation that don't affect optimal warping and transform necessary portion of "warped space" to device normalized coordinates. On the other hand projection vector is just fourth column of a transformation matrix (D3D-like row-vector notation used through this paper).

Let's take arbitrary projection vector (without loose of generality we can always divide by fourth component) this vector define R^3 optimization space, but after application of necessary constraints (every point of a light ray should project to the same shadow map point) dimension reduce to R^2 parameterization space with simple analytical structure, that helped to figure out suboptimal solution and completely resolve Z bias non-linearity issue.

2. General case analysis

2.1 Parameterization plane

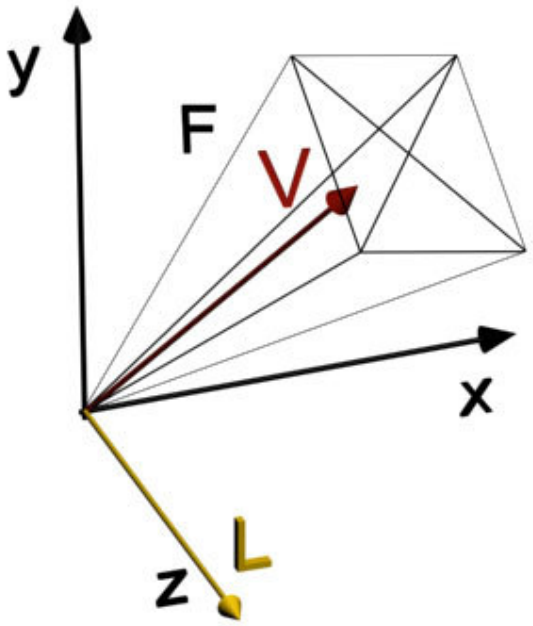


Figure 2: *An example of the light space with the light direction L, view vector V and view frustum F.*

In the light space the light direction is parallel to Z axis and the viewer origin translated to $(0,0,0)$ .Any other case can be reduced into this one. Let's assume an arbitrary transformation M[1] is applied to the light space. For any perspective parameterization it would be necessary that each point of a light ray is projected in the same point (X',Y') of a shadow map. In other words the point (X',Y') should not depend on the free variable t, thus the following constraints should be satisfied $Cw = 0$ and $\vec{C}xy = (0,0)$.

$$(X',Y') = (X,Y,t,1)\cdot M = (X,Y,t,1)\cdot \begin{pmatrix} Ax & Ay & Az & Aw \\ Bx & By & Bz & Bw \\ Cx & Cy & Cz & Cw \\ Dx & Dy & Dz & 1 \end{pmatrix} =$$

$$= \frac{\vec{A}xy\cdot X + \vec{B}xy\cdot Y + \vec{C}xy\cdot t + \vec{D}xy}{Aw\cdot X + Bw\cdot Y + Cw\cdot t + 1} \quad (1)$$

The basic formula of arbitrary parameterization:

$$Cw = 0, \vec{C}xy = (0,0)$$

$$(X',Y') = (\vec{A}xy\cdot X + \vec{B}xy\cdot Y + \vec{D}xy)\cdot \frac{1}{Aw\cdot X + Bw\cdot Y + 1} \quad (2)$$

Then the projection vector for arbitrary parameterization is $(Aw, Bw, 0, 1)$ where $(Aw, Bw) \in R^2$ is a pair of free variables defining a *parameterization plane* which is equivalent to the xy plane. Thus determining of parameterization is reduced to fixing of two free variables in the xy plane. In polar coordinates this is equivalent to fixing of the projection vector direction and length.

2.2 Back projection singularity

It's easy to see that the perspective part of (2) is equivalent to

$$\frac{1}{Aw\cdot X + Bw\cdot Y + 1} = \frac{1}{k+1}$$

where k is the length of projection of a vector (X,Y) (where X, Y are the coordinates of light ray) to some vector of free variables $\vec{P}\ (Aw, Bw)$ in the xy plane. This projection consists of two intervals, the *view frustum interval* $k \in (0..FarPlane)$ for light rays that intersect frustum and the *back projection interval* $k \in (-1..0)$ for light rays that intersect shadow casters behind a viewer.

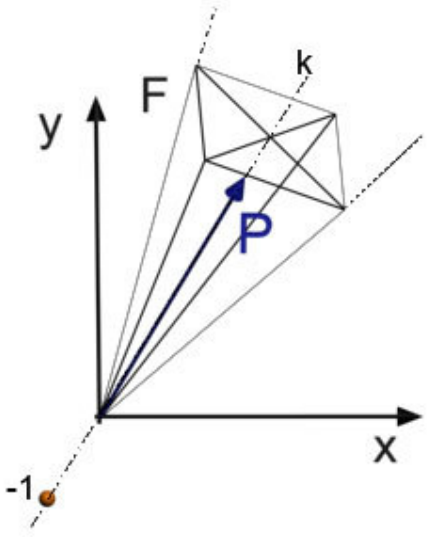


Figure 3: *Frustum projection to the xy plane with a vector of free variables P.*

There is a singularity for $k = -1$, thus all light rays behind the viewer should be in the back projection interval $k \in (-1..0)$. In cases when shadow casters are located far behind the viewer, the only way is to scale the back projection interval into $k \in (-1..0)$ range using very small projection vector $\vec{P}$. But this compression affects view frustum interval too, which leads to significant degradation of shadows quality in front of the viewer. Every PSM method in some way should handle casters behind a viewer and therefore is subject to the given problem.

[1] without loose of generality we can always divide by Dw that is just change of a unit scale

3. Design

3.1 Projection vector direction

Distribution of light rays in a view frustum xy projection is dense and random hence there is no preferable direction. Thus the only balanced choice is the direction with *symmetric* warping effect for the whole frustum. The projection of the view frustum to xy plane can be in most cases bounded by the triangle with central line of symmetry having the same direction as projection of view direction $\vec{V}$ to xy plane. Then the projection vector direction is:

$$unit\vec{P} = \frac{\vec{V}xy}{|\vec{V}xy|}.$$

3.2 Optimal warping effect

In ideal case the warping effect should depend only on the distance to the viewer origin. That is near objects have high shadow map resolution and far objects low. Thus in optimal case warping effect should not depend on relative viewer and light position\direction.

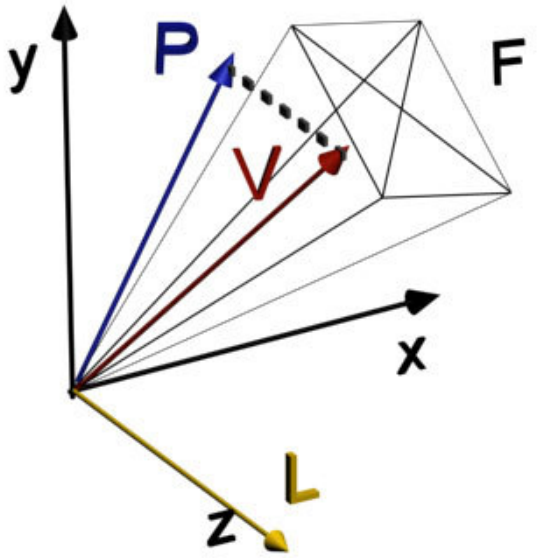


Figure 4: *The view frustum F, the light direction L and P the view vector V projection to xy plane.*

Let's take some point on a view vector line with distance *d* from the viewer origin and *gamma* an angle between the view vector $\vec{V}$ and its projection $\vec{P}$ to xy plane, from (2):

$$\frac{1}{Aw \cdot X + Bw \cdot Y + 1} = \frac{1}{\cos(gamma) \cdot d \cdot lengthP + 1}$$

Then for optimal warping effect

$$lengthP = \frac{coef}{\cos(gamma)} \text{ and}$$

$$\frac{1}{\cos(gamma) \cdot d \cdot \left(\frac{coef}{\cos(gamma)}\right) + 1} = \frac{1}{d \cdot coef + 1}$$

where *coef* – constant tuned for scene scale and desirable trade-off between near\far objects shadow map resolution.

3.3 Handling of back projection singularity

For correct back projection every point of a back projection interval should be in $k \in (-1..0)$ range. Then following algorithm computes optimal bound of a projection vector length:

1. For all points in *Casters*[2] find the minimal length of projection to $unit\vec{P}$, we'll refer to it as *minCastersProj*.

$$minCastersProj = \min[for\ all\ \ p\ in\ Casters]\ (px \cdot unit\vec{P}x + py \cdot unit\vec{P}y)$$

2. For all points in *Receivers*[2] find the minimal length of projection to $unit\vec{P}$, we'll refer to it as *minReceiversProj*.

$$minReceiversProj = \min[for\ all\ \ p\ in\ Receivers]\ (px \cdot unit\vec{P}x + py \cdot unit\vec{P}y)$$

3. The focus region interval is the intersection of the *Casters* and *Receivers* intervals. Then the minimal point of the focus region interval is:

$$minFocusRegionProj = \max(minCastersProj, minReceiversProj)$$

4. Calculate the projection vector maximal length.

$$(unit\vec{P}x \cdot X + unit\vec{P}y \cdot Y) \cdot lengthP + 1 = W$$

$$minFocusRegionProj \cdot maxLengthP + 1 = epsilonW$$

$$maxLengthP = \frac{epsilonW - 1}{\max(minCastersProj, minReceiversProj)} \quad (3)$$

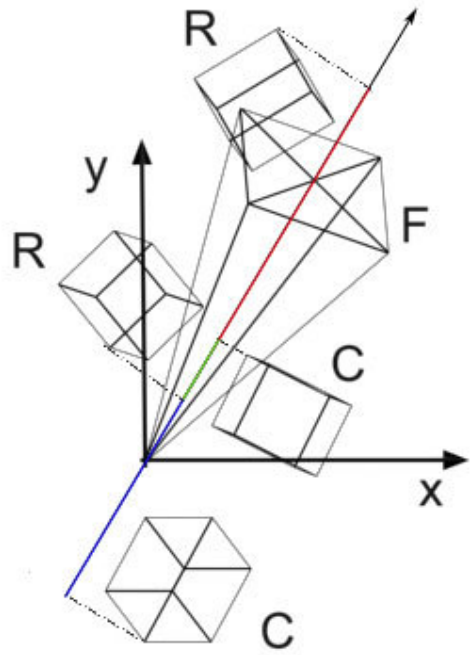


Figure 5: *R – receiver, C – caster, red and blue line is projection intervals of receivers and casters with focus region interval marked by green.*

[2] assume that casters and receivers transformed into light space

3.4 Minimization of shadow map waste

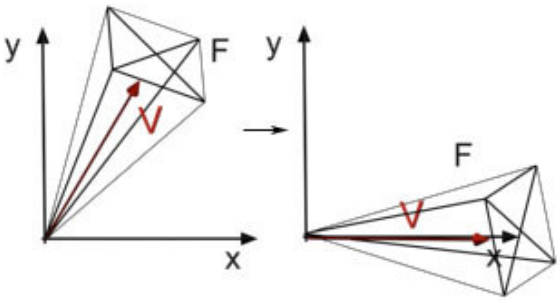


Figure 6: *PPLS before (left) and after (right) applying rotation*

Rotation around Z axis in post projection light space (PPLS) is applied to minimize a shadow map waste. This transformation aligns the projection vector direction with an X axis producing optimal PPLS AABB's.

$$ZRotation = \begin{pmatrix} unit\vec{P}x & unit\vec{P}y & 0 & 0 \\ unit\vec{P}y & -unit\vec{P}x & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3.5 Post projection light space

Clipping of length of the projection vector is applied to avoid singularity:
if $(maxLengthP > 0$ and $lengthP > maxLengthP)$
then $lengthP = maxLengthP$

After fixing two free variables the projection vector is $(unit\vec{P}x \cdot lengthP, unit\vec{P}y \cdot lengthP, 0, 1)$. And a projective transformation:

$$Projection = \begin{pmatrix} 1 & 0 & 0 & unit\vec{P}x \cdot lengthP \\ 0 & 1 & 0 & unit\vec{P}y \cdot lengthP \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Then transformation from the light space to the post projection light space (PPLS) is:

$$PPLS\ Transform = Projection * ZRotation .$$

3.6 Focus region

Let's define *Receivers* as set of points of shadow receivers bounding volumes and *Casters* as set of points of shadow casters bounding volumes. Then *PPLS Receivers, PPLS Casters* is bounding volumes points transformed to the post projection light space.

PPLS Receivers AABB and *PPLS Casters AABB* is two axes aligned bounding boxes (AABB) constructed in the post projection light space (PPLS) from *PPLS Receivers* and *PPLS Casters*, respectively.

$$PPLS\ Receivers = Receivers[for\ all\ points] * LightSpace * PPLS\ Transform$$
$$PPLS\ Casters = Casters[for\ all\ points] * LightSpace * PPLS\ Transform$$

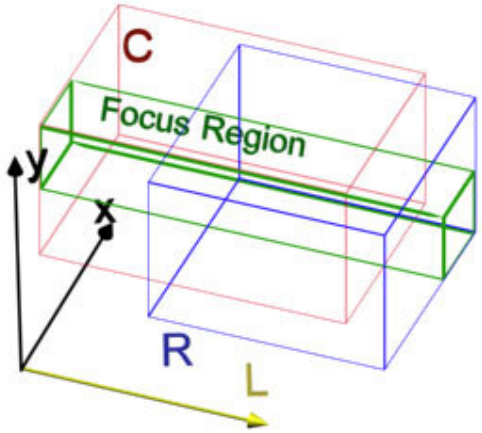


Figure 7: *R – AABB of receivers in PPLS, C – AABB of casters in PPLS*

Then the *focus region* is geometrical intersection in xy plane of a *PPLS Receivers AABB* and a *PPLS Casters AABB*. Thus the focus region bounds a set of light rays which intersect at least one a shadow caster AND a shadow receiver.

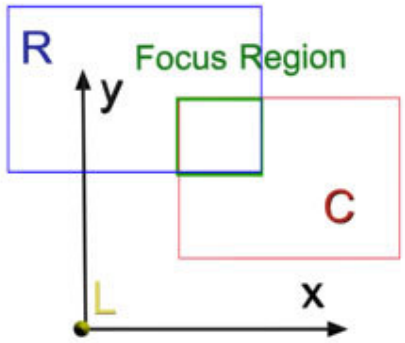


Figure 8: *R – AABB of receivers in PPLS, C – AABB of casters in PPLS*

3.7 Focus region basis

Having the focus region it is possible to construct linear basis in the post projection light space that transform the focus region into a unit cube and doesn't affect optimal warping. Let's assume that the focus region box in the post projection light space is defined by the maximal *MaxPointXY* and the minimal *MinPointXY* points. To construct basis, Z range $(MinZ, MaxZ)$ for the focus region calculated. Where *MinZ* is minimal Z coordinate of *Receivers* and *Casters* post projection light space axis aligned bounding boxes and *MaxZ* is maximal Z coordinate.

ZMin = min(*PPLS Casters AABB MinPoint Z*,
PPLS Receivers AABB MinPoint Z)
ZMax = max(*PPLS Casters AABB MaxPoint Z*,
PPLS Receivers AABB MaxPoint Z)

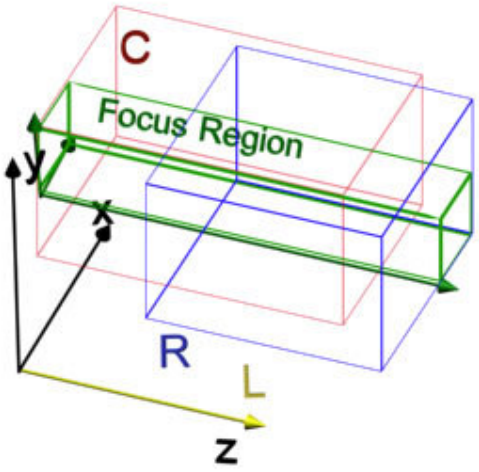


Figure 9: *R – AABB of receivers in PPLS, C – AABB of casters in PPLS and the focus region basis marked by green*

Thus the focus region basis that includes all necessary PPLS points can be calculated as:

$$FocusRegionBasis = \begin{pmatrix} MaxPointX - MinPointX & 0 & 0 & 0 \\ 0 & MaxPointY - MinPointY & 0 & 0 \\ 0 & 0 & MaxZ - MinZ & 0 \\ MinPointX & MinPointY & MinZ & 1 \end{pmatrix}$$

3.8 Unit cube

The focus region transformation into a unit cube is just matrix inversion of the focus region basis.

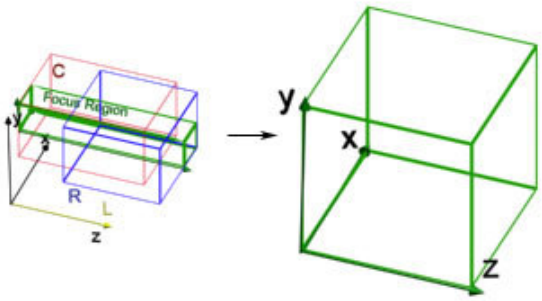


Figure 10: *PPLS (left) and the focus region basis transformed into a unit cub (right)*

$$UnitCubeTransform = FocusRegionBasis^{-1}$$

$$UnitCubeTransform = \begin{pmatrix} \frac{1}{MaxPointX - MinPointX} & 0 & 0 & 0 \\ 0 & \frac{1}{MaxPointY - MinPointY} & 0 & 0 \\ 0 & 0 & \frac{1}{MaxZ - MinZ} & 0 \\ -\frac{MinPointX}{MaxPointX - MinPointX} & -\frac{MinPointY}{MaxPointY - MinPointY} & -\frac{MinZ}{MaxZ - MinZ} & 1 \end{pmatrix}$$

3.9 Final transformation

Finally total transformation can be easily computed as:

$$XPSMTransform = View * LightSpace * PPLSTransform * UnitCubeTransform * NormalizedSpace$$

where
View – transformation into the camera view space,
LightSpace – transformation into the light space,
PPLS Transform – transformation from the light space into the post projection light space,
UnitCubeTransform – transformation of the focus region basis in the post projection light space into a unit cube,
NormalizedSpace – transformation from the unit cube into Direct3D\OpenGL normalized device coordinates space.

3.10 Correct Z bias

Let's take some point $\vec{P}$ transform it into the light space, apply Z bias and following XPSM transformations, then:

$$\vec{P}biased = (\vec{P} * LightSpace - (0,0,bias,0)) * * PPLSTransform * UnitCubeTransform * * NormalizedSpace =$$

$$\vec{P} * LightSpace * PPLSTransform * UnitCubeTransform * NormalizedSpace - (0,0,bias,0) * PPLSTransform * UnitCubeTransform * * NormalizedSpace =$$

$$\vec{P} * XPSMTransform - (0,0,\frac{bias}{MaxZ - MinZ},0)$$

And finally:

$$Z' = \frac{Z - ZBias}{W},$$ where *ZBias* is

$$ZBiasD3D = \frac{bias}{MaxZ - MinZ}$$

$$ZBiasOGL = \frac{2 \cdot bias}{MaxZ - MinZ}$$

(due to different normalized device coordinates)

3.11 Miner's lamp case

For the case of miner's lamp, we just turn to uniform shadow maps, which is known to be ideal for this case.

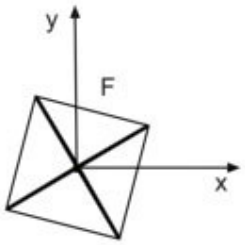


Figure 11: *Miner's lamp case*

4. Algorithm

Input: *Receivers* and *Casters* – the set of shadow receivers and casters bounding volumes points in the *view space,* L – the light direction in the *view space*, *epsilonW* – constant defining how far to "infinity", warping effect can be pushed in critical case, *coef* – constant tuned for scene scale and desirable trade-off between near\far objects shadow map resolution.

Step 1 Compute the light direction look-at matrix to transform from a view space into the light space.

$$LightSpace = LookAt((0,0,0), \vec{L})$$ (***left-handed version***)

Step 2 Transform the view vector into the light space.

$$\vec{V} = (0,0,1) * LightSpace$$

Step 3 Project the view vector into the parameterization, xy plane and normalize the projection vector.

$$unit\vec{P} = \frac{\vec{V}xy}{\left|\vec{V}xy\right|}$$

if ($\left|\vec{V}xy\right| < 0.1$) then miner lamp case, use orthogonal shadows with focusing.

Step 4 Transform *Casters* and *Receivers* into the light space, project bounding volumes points into $unit\vec{P}$ and find projections with minimal length.

$$minCastersProj = \min[for\ all\ p\ in\ LSCasters](px \cdot unit\vec{P}x + py \cdot unit\vec{P}y)$$

$$minReceiversProj = \min[for\ all\ p\ in\ LSReceivers](px \cdot unit\vec{P}x + py \cdot unit\vec{P}y)$$

Step 5 Calculate maximum length of the projection vector.

$$maxLengthP = \frac{epsilonW - 1}{\max(minCastersProj, minReceiversProj)}$$

Step 6 Calculate optimal length of the projection vector.

gamma – angle between $\vec{P}$ and $\vec{V}$.

$$cosGamma = \vec{V}x \cdot unit\vec{P}x + \vec{V}y \cdot unit\vec{P}y$$

$$lengthP = \frac{coef}{cosGamma}$$

Step 7 Clip the projection vector length.

if (*maxLengthP* > 0 and *lengthP* > *maxLengthP*)
then *lengthP* = *maxLegthP*

Step 8 Calculate the projection matrix.

$$Projection = \begin{pmatrix} 1 & 0 & 0 & unit\vec{P}x \cdot lengthP \\ 0 & 1 & 0 & unit\vec{P}y \cdot lengthP \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step 9 Calculate basis of rotation around Z axis.

$$ZRotation = \begin{pmatrix} unit\vec{P}x & unit\vec{P}y & 0 & 0 \\ unit\vec{P}y & -unit\vec{P}x & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step 10 Compute transformation from the light space into the post projection light space.

$$PPLS\ Transform = Projection * ZRotation$$

Step 11 Transform *Casters* and *Receivers* points into the post projection light space (PPLS).

$$PPLS\ Receivers = Receivers[for\ all\ points] * LightSpace * PPLS\ Transform$$

$$PPLS\ Casters = Casters[for\ all\ points] * LightSpace * PPLS\ Transform$$

Before final division clip points with w < epsilonW

Step 12 Find AABB's in the post projection light space for *Casters* and *Receivers*.

PPLS CastersBox = Build AABB (*PPLS Casters*)
PPLS ReceiversBox = Build AABB (*PPLS Receivers*)

Step 13 Find the focus region that is intersection of the *PPLS CastersBox* and *PPLS ReceiversBox* in xy plane.

FocusRegion = Intersect 2D (*PPLS Casters AABB*, *PPLS Receivers AABB*)

Step 14 Calculate the focus region basis.

ZMin = min(*PPLS Casters AABB MinPoint Z*, *PPLS Receivers AABB MinPoint Z*)
ZMax = max(*PPLS Casters AABB MaxPoint Z*, *PPLS Receivers AABB MaxPoint Z*)

$$FocusRegionBasis = \begin{pmatrix} MaxPointX - MinPointX & 0 & 0 & 0 \\ 0 & MaxPointY - MinPointY & 0 & 0 \\ 0 & 0 & MaxZ - MinZ & 0 \\ MinPoint\ X & MinPoint\ Y & MinZ & 1 \end{pmatrix}$$

Step 15 Calculate inversion of the focus region basis that is transformation into a unit cube.

$$UnitCubeTransform = FocusRegionBasis^{-1}$$

$$UnitCubeTransform = \begin{pmatrix} \frac{1}{MaxPointX - MinPointX} & 0 & 0 & 0 \\ 0 & \frac{1}{MaxPointY - MinPointY} & 0 & 0 \\ 0 & 0 & \frac{1}{MaxZ - MinZ} & 0 \\ -\frac{MinPoint\ X}{MaxPointX - MinPointX} & -\frac{MinPoint\ Y}{MaxPointY - MinPointY} & -\frac{MinZ}{MaxZ - MinZ} & 1 \end{pmatrix}$$

Step 16 Calculate the transformation from the unit cube into the device normalized coordinates space.

$$NormalizedSpaceD3D = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & 0 & 1 \end{pmatrix}$$

$$NormalizedSpaceOGL = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ -1 & -1 & -1 & 1 \end{pmatrix}$$

Step 17 Compute final transformation.

$$XPSM\ Transform = View * LightSpace * PPLS\ Transform * UnitCubeTransform * NormalizedSpace$$

Step 18 Calculate correct Z bias.

Shader code:

$$Z' = \frac{Z - ZBias}{W},$$

where *ZBias* is

$$ZBiasD3D = \frac{bias}{MaxZ - MinZ}$$

$$ZBiasOGL = \frac{2 \cdot bias}{MaxZ - MinZ}$$

Output: *XPSM Transform*, *ZBias*

4. Results and conclusion

XPSM was successfully implemented as a part of NVIDIA PracticalPSM demo (can be downloaded from http://xpsm.org). In average XPSM perform better than PSM/LiSPSM and shadows resolution directly comparable to TSM scenes. More important, that in worst cases XPSM consistently outperforms PSM/LiSPSM/TSM and never produces any significant visual artifacts (Z acne, etc) or singularities. These characteristics make XPSM a very practical approach for shadow map parameterization.

If you find this research and development efforts valuable or want to support, please, make donation.
http://donate.xpsm.org

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

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

[7] Birchfield. S. An Introduction to Projective Geometry (for computer vision). http://vision.stanford.edu/~birch/projective/projective.pdf

[8] Davis T. Projective Geometry http://www.geometer.org/mathcircles/projective.pdf

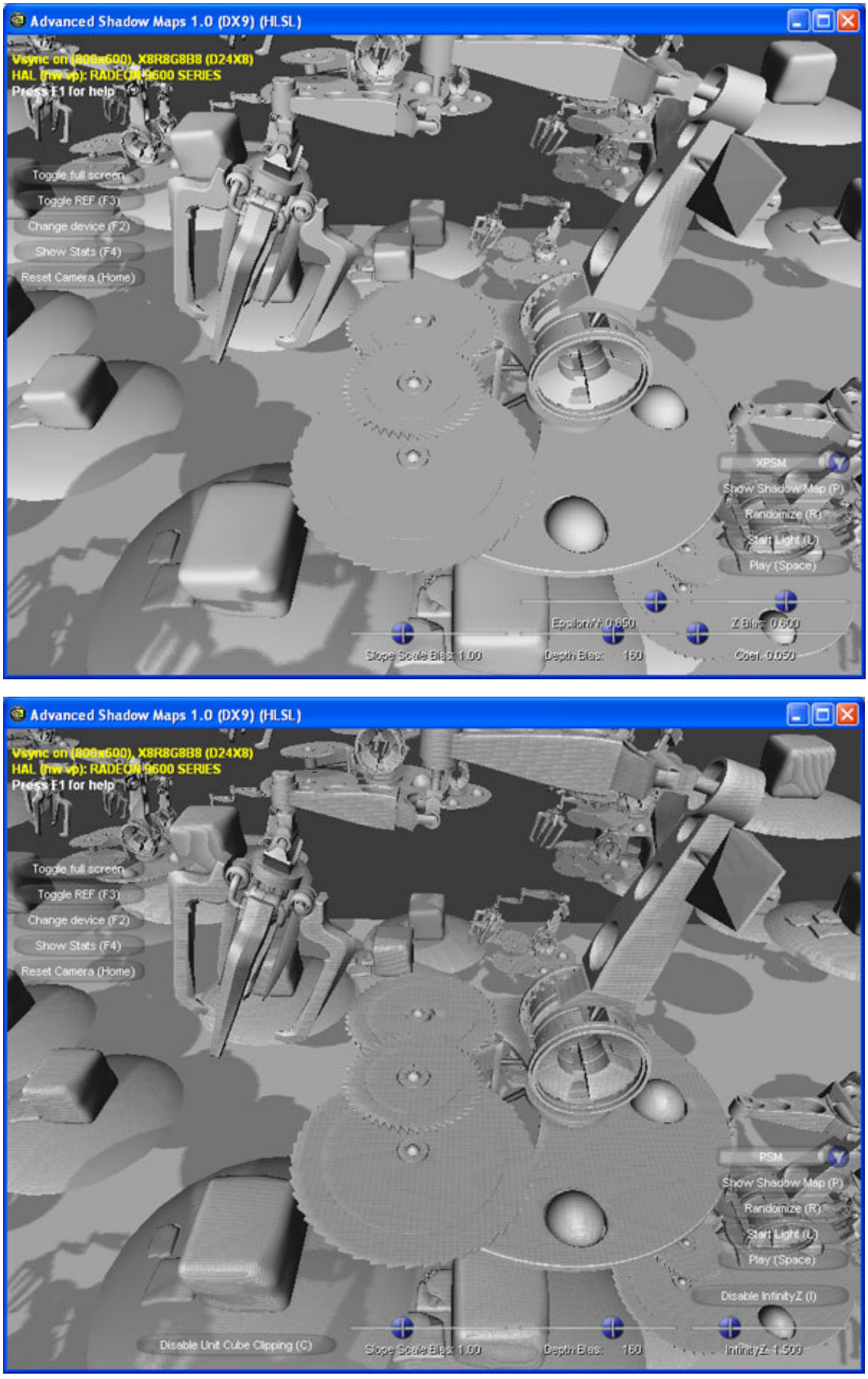


Figure 12a: *XPSM (top), PSM(bottom)*

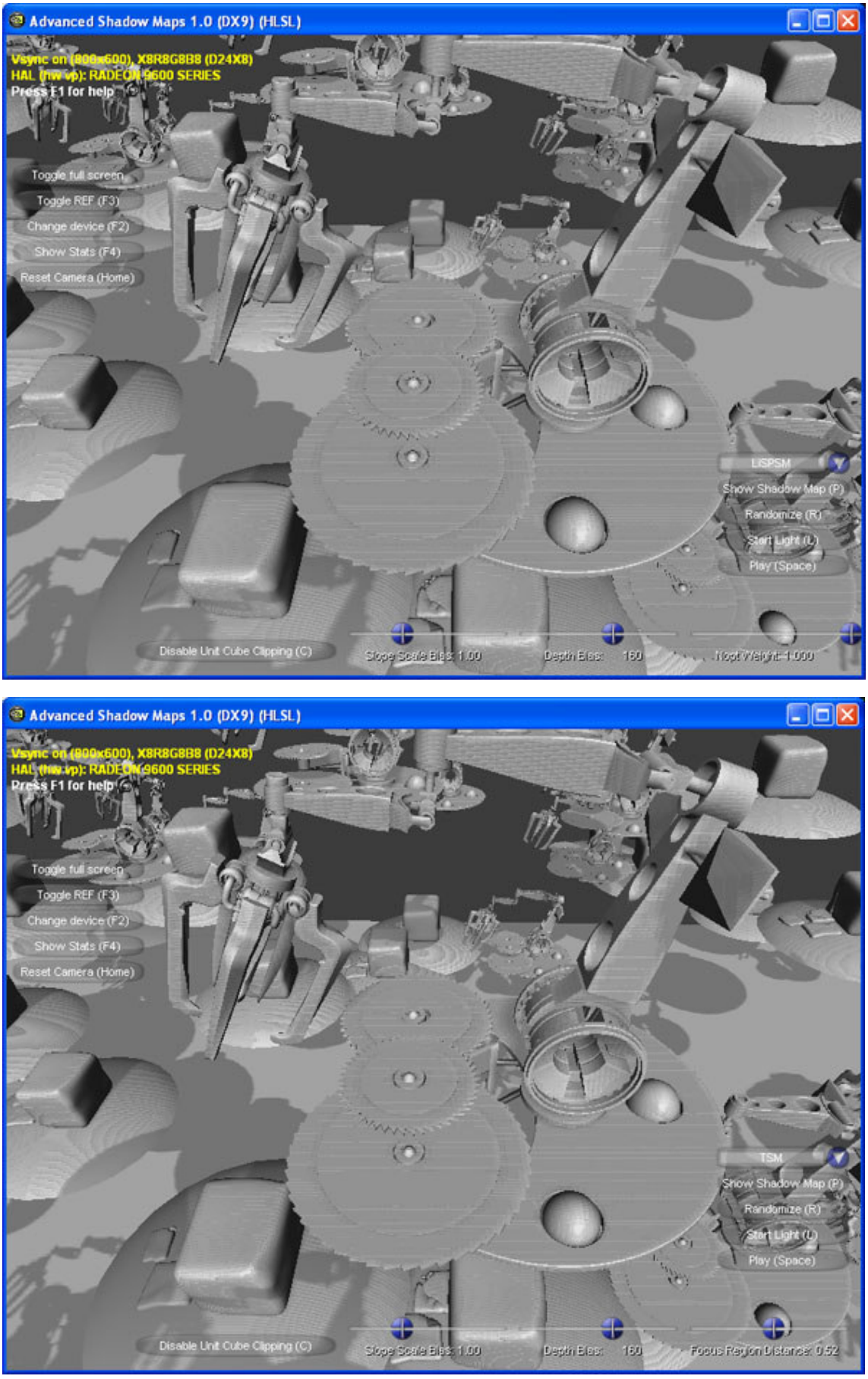


Figure 12b: LiSPSM (top), TSM(bottom)

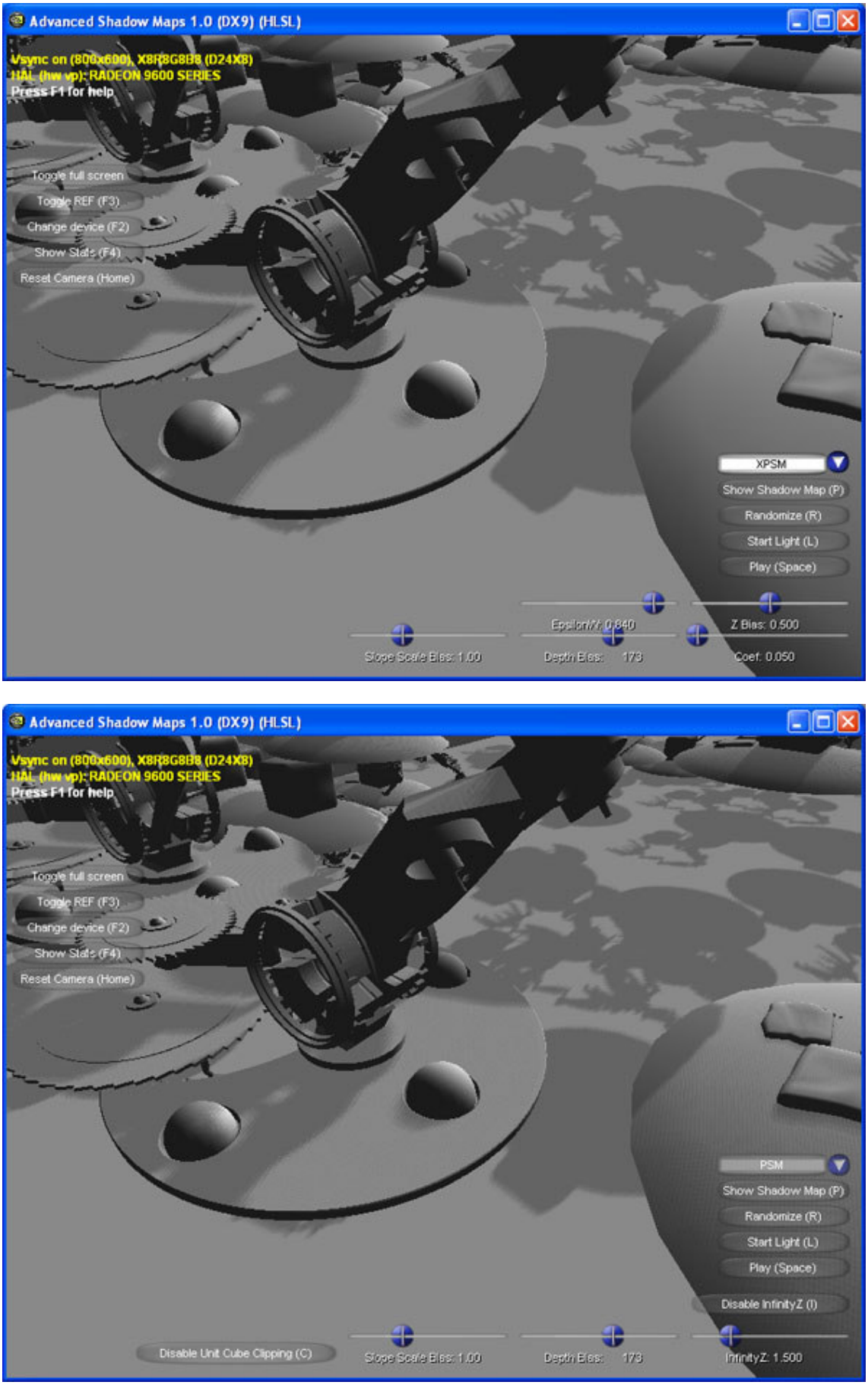
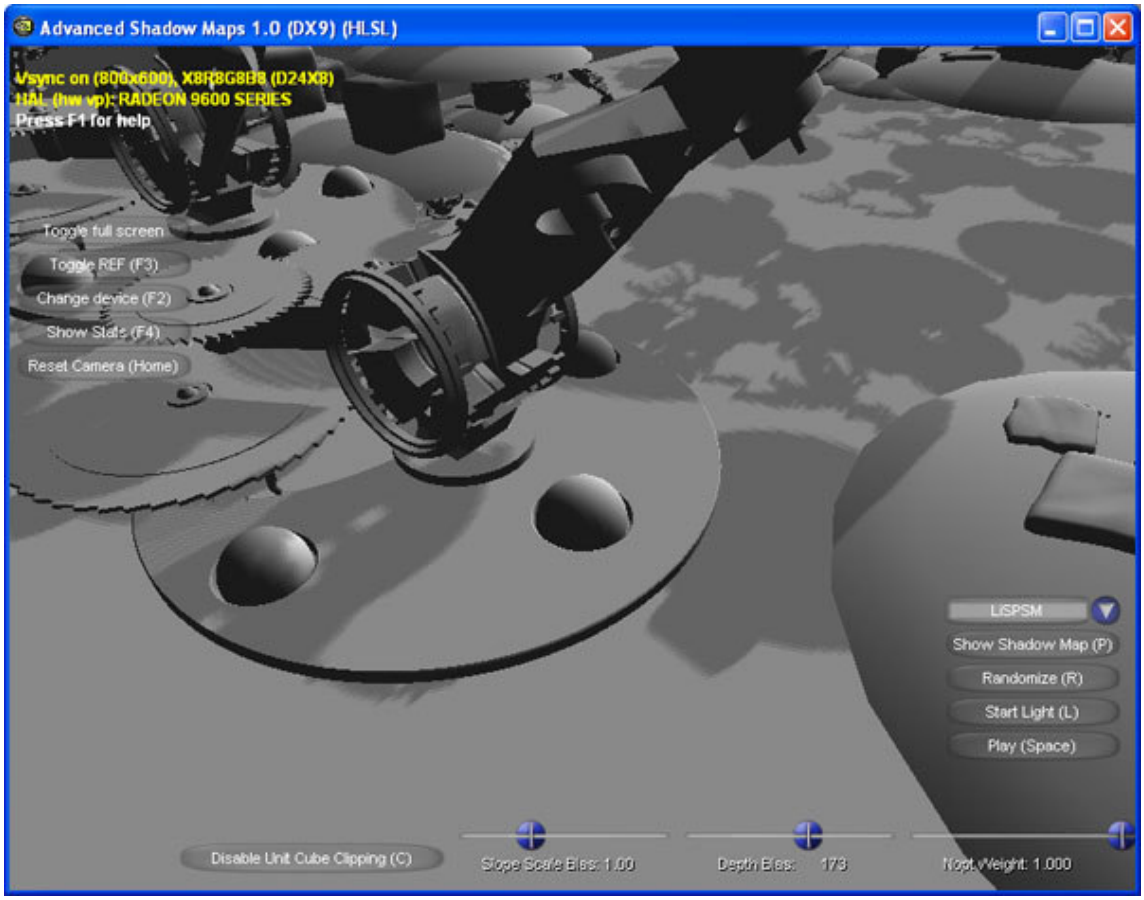


Figure 13a: *XPSM(top), PSM(bottom)*



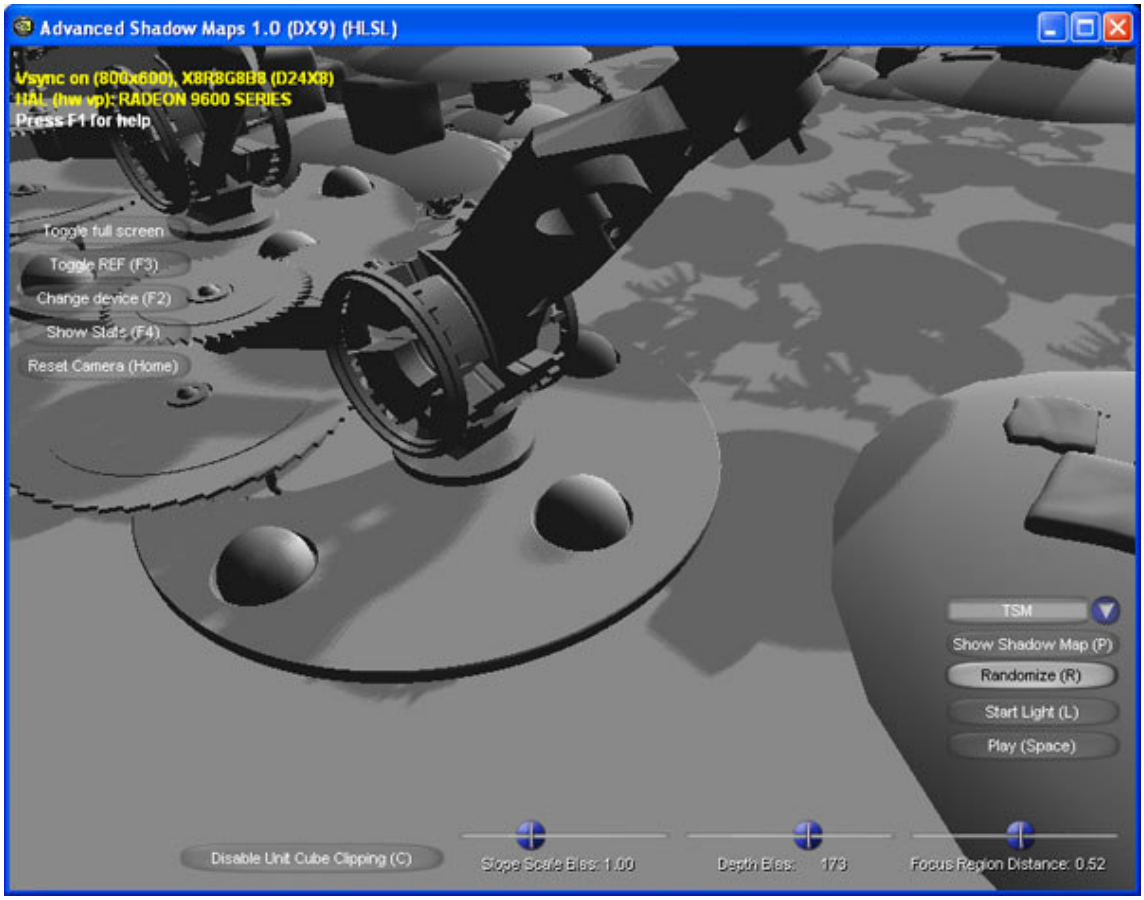


Figure 13b: *LiSPSM(top), TSM(bottom)*