Perspective Correct Interpolation

For

Dual Paraboloid Mapping

This file is a document to explain how to do the Perspective Correct Interpolation correctly for the Dual Paraboloid Mapping.

目录：

透视投影

Reversed-Z

Nathan Reed. "Depth Precision Visualized." NVIDIA GameWorks Blog 2015.

https://developer.nvidia.com/content/depth-precision-visualized

与本文无关

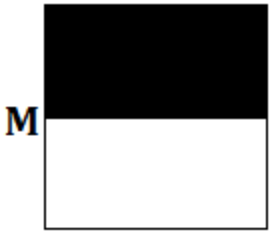
透视投影的透视校正插值

Jim Blinn. "Hyperbolic Interpolation." IEEE 1992.
https://doi.ieeecomputersociety.org/10.1109/MCG.1992.10028

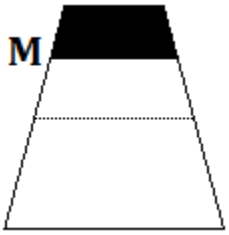
在 Direct3D12、Vulkan 中默认在插值时透视校正（由硬件完成）
HLSL/Reference for HLSL/Language Syntax/Variables/Data Types/InterpolationModifier
http://msdn.microsoft.com/en-us/library/windows/desktop/bb509668
GLSL Specification/Variables and Types/Interpolation Qualifiers
http://www.khronos.org/registry/OpenGL/index_gl.php

在 Vulkan Specification 中给出了透视校正插值的公式
直线图元
Rasterization/Line Segments/"perspective interpolation"
三角形图元
Rasterization/Polygons/"perspective interpolation"

比如我们有一个黑白相间的正方形，M 恰好为正方形的边上的中点（可以认为是 UV = 0.5 的点）：



我们假定该正方形经过透视投影映射到二维空间中的梯形：



我们发现，根据“近大远小”的原理，黑白分界线会向远处偏移，在二维空间中，M（UV = 0.5 的点）不再是梯形的边上的中点。因此，图形流水线的光栅化阶段在插值计算各个像素的 UV 时，不能简单地按照二维空间中的比例（即 λ_normdev）进行计算。

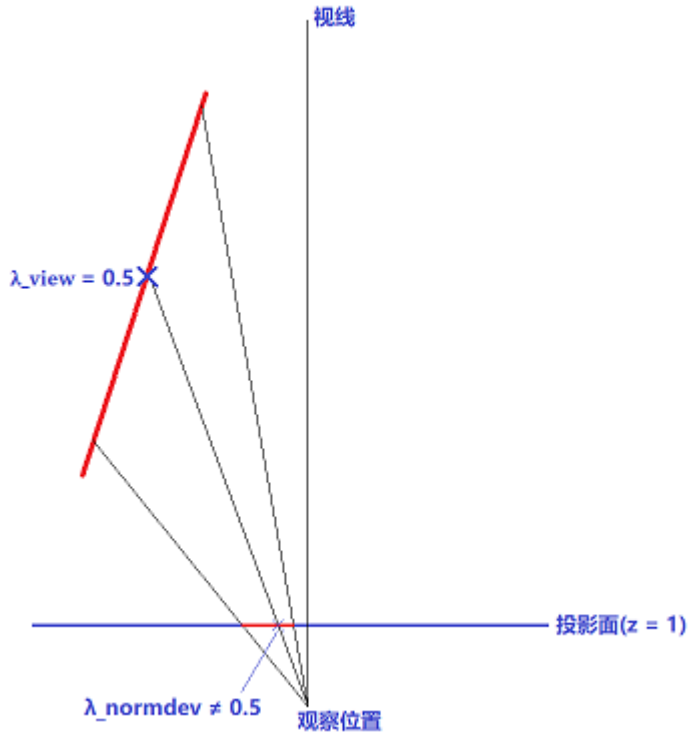
我们回忆透视投影中三维空间中的几何体顶点经过 MVP 变换映射到二维空间中的过程：

ModelSpace->WorldSpace->ViewSpace

$$\begin{bmatrix} x\_clip \\ y\_clip \\ z\_clip \\ w\_clip \end{bmatrix} = PVM \begin{bmatrix} x\_model \\ y\_model \\ z\_model \\ 1 \end{bmatrix}$$ //w_clip = z_view

ClipSpace -> NormalizedDeviceSpace

$$\begin{bmatrix} x\_normdev \\ y\_normdev \\ z\_normdev \\ 1 \end{bmatrix} = \frac{1}{w\_clip} \begin{bmatrix} x\_clip \\ y\_clip \\ z\_clip \\ w\_clip \end{bmatrix}$$ //由图形流水线中的固定功能阶段完成



我们设根据△ABC 插值得到点 P：

$$\begin{bmatrix} P.x\_model/world/view \\ P.y\_model/world/view \\ P.z\_model/world/view \end{bmatrix} = \lambda A\_model/world/view \begin{bmatrix} A.x\_model/world/view \\ A.y\_model/world/view \\ A.z\_model/world/view \end{bmatrix} + \lambda B\_model/world/$$

$$view \begin{bmatrix} B.x\_model/world/view \\ B.y\_model/world/view \\ B.z\_model/world/view \end{bmatrix} + \lambda C\_model/world/view \begin{bmatrix} C.x\_model/world/view \\ C.y\_model/world/view \\ C.z\_model/world/view \end{bmatrix}$$

$$\begin{bmatrix} P.x\_normdev \\ P.y\_normdev \end{bmatrix} = \lambda A\_normdev \begin{bmatrix} A.x\_normdev \\ A.y\_normdev \end{bmatrix} + \lambda B\_normdev \begin{bmatrix} B.x\_normdev \\ B.y\_normdev \end{bmatrix} + \lambda C\_normdev \begin{bmatrix} C.x\_normdev \\ C.y\_normdev \end{bmatrix}$$

由于 ModelSpace->WorldSpace->ViewSpace 是线性变换，不难证明 λ_view = λ_world = λ_model。但是 ViewSpace->ClipSpace->NormalizedDeviceSpace 并不是线性变换，导致了二维空间中的 λ_normdev 不再与三维空间中的 λ_view 相等，从而产生了上述问题。

我们只要找到某种方式，基于 λ_normdev 求出 λ_view，就可以做到按照三维空间中的比例 λ_model 对顶点属性 VTXATTR 进行插值。

Blinn 在 1992 年提出了用双曲（Hyperbolic）插值来解决这个问题，这也是目前图形流水线中的光栅化阶段所使用的方法，被称为透视校正（Perspective Correct）插值（由于双抛物面映射的透视校正插值由本人原创，在没有特殊说明的情况下，透视校正插值即指透视投影的透视校正插值）：

$$P.VTXATTR = \frac{\lambda A\_normdev\frac{1}{A.w\_clip}A.VTXATTR + \lambda B\_normdev\frac{1}{B.w\_clip}B.VTXATTR + \lambda C\_normdev\frac{1}{C.w\_clip}C.VTXATTR}{\lambda A\_normdev\frac{1}{A.w\_clip} + \lambda B\_normdev\frac{1}{B.w\_clip} + \lambda C\_normdev\frac{1}{C.w\_clip}} \quad //w\_clip = z\_view$$

//VTXATTR 可以是任意顶点属性，包括但不限于 UV

可以证明：

$$\lambda A\_hw = \frac{\lambda A\_normdev\frac{1}{A.w\_clip}}{\lambda A\_normdev\frac{1}{A.w\_clip} + \lambda B\_normdev\frac{1}{B.w\_clip} + \lambda C\_normdev\frac{1}{C.w\_clip}} = \lambda A\_view = \lambda A\_world = \lambda A\_model$$

$$\lambda B\_hw = \frac{\lambda B\_normdev\frac{1}{B.w\_clip}}{\lambda A\_normdev\frac{1}{A.w\_clip} + \lambda B\_normdev\frac{1}{B.w\_clip} + \lambda C\_normdev\frac{1}{C.w\_clip}} = \lambda B\_view = \lambda B\_world = \lambda B\_model$$

$$\lambda C\_hw = \frac{\lambda C\_normdev\frac{1}{C.w\_clip}}{\lambda A\_normdev\frac{1}{A.w\_clip} + \lambda B\_normdev\frac{1}{B.w\_clip} + \lambda C\_normdev\frac{1}{C.w\_clip}} = \lambda C\_view = \lambda C\_world = \lambda C\_model$$

证明：

我们设三维空间中共面的四点 P、A、B、C 在 ViewSpace 中的坐标为$\overrightarrow{OP_{View}}$、$\overrightarrow{OA_{View}}$、$\overrightarrow{OB_{View}}$、$\overrightarrow{OC_{View}}$

根据 λ_normdev 的定义，我们有：

$$\begin{bmatrix} P.x\_normdev \\ P.y\_normdev \end{bmatrix} = \lambda A\_normdev \begin{bmatrix} A.x\_normdev \\ A.y\_normdev \end{bmatrix} + \lambda B\_normdev \begin{bmatrix} B.x\_normdev \\ B.y\_normdev \end{bmatrix} + \lambda C\_normdev \begin{bmatrix} C.x\_normdev \\ C.y\_normdev \end{bmatrix}$$

上式也可以看作三维空间中的坐标：

$$\begin{bmatrix} P.x\_normdev \\ P.y\_normdev \\ 1 \end{bmatrix} = \lambda A\_normdev \begin{bmatrix} A.x\_normdev \\ A.y\_normdev \\ 1 \end{bmatrix} + \lambda B\_normdev \begin{bmatrix} B.x\_normdev \\ B.y\_normdev \\ 1 \end{bmatrix} + \lambda C\_normdev \begin{bmatrix} C.x\_normdev \\ C.y\_normdev \\ 1 \end{bmatrix}$$

即：

$$\begin{bmatrix} P.x\_view \\ P.y\_view \\ P.z\_view \end{bmatrix} = \lambda A\_normdev \frac{1}{A.z\_view} \begin{bmatrix} A.x\_view \\ A.y\_view \\ A.z\_view \end{bmatrix} + \lambda B\_normdev \frac{1}{B.z\_view} \begin{bmatrix} B.x\_view \\ B.y\_view \\ B.z\_view \end{bmatrix} + \lambda C\_normdev \frac{1}{C.z\_view} \begin{bmatrix} C.x\_view \\ C.y\_view \\ C.z\_view \end{bmatrix}$$

即：

$$\overrightarrow{OP_{View}} = \lambda A\_normdev \frac{1}{A.z\_view} \overrightarrow{OA_{View}} + \lambda B\_normdev \frac{1}{B.z\_view} \overrightarrow{OB_{View}} + \lambda C\_normdev \frac{1}{C.z\_view} \overrightarrow{OC_{View}}$$

由于 P、A、B、C 四点共面，根据共面向量定理，我们有：
存在唯一的实数 λA_view、λB_view、λC_view，使$\overrightarrow{OP_{View}} = \lambda A_view\overrightarrow{OA_{View}} + \lambda B_view\overrightarrow{OB_{View}} + \lambda C_view\overrightarrow{OC_{View}}$，且 λA_view + λB_view + λC_view = 1；

由于唯一性，我们有：

$$\lambda A\_view = \lambda A\_normdev \frac{1}{A.z\_view}$$

$$\lambda B\_view = \lambda B\_normdev \frac{1}{B.z\_view}$$

$$\lambda C\_view = \lambda C\_normdev \frac{1}{C.z\_view}$$

$$\lambda A\_normdev \frac{1}{A.z\_view} + \lambda B\_normdev \frac{1}{B.z\_view} + \lambda C\_normdev \frac{1}{C.z\_view} = 1$$

由于 w_clip = z_view，我们有：

$$\lambda A\_hw = \frac{\lambda A\_normdev_{\frac{1}{A.w\_clip}}}{\lambda A\_normdev_{\frac{1}{A.w\_clip}} + \lambda B\_normdev_{\frac{1}{B.w\_clip}} + \lambda C\_normdev_{\frac{1}{C.w\_clip}}} =$$

$$\frac{\lambda A\_normdev_{\frac{1}{A.w\_vew}}}{\lambda A\_normdev_{\frac{1}{A.w\_view}} + \lambda B\_normdev_{\frac{1}{B.w\_view}} + \lambda C\_normdev_{\frac{1}{C.w\_view}}} = \frac{\lambda A\_view}{1} = \lambda A\_view$$

$$\lambda B\_hw = \frac{\lambda B\_normdev_{\frac{1}{A.w\_clip}}}{\lambda A\_normdev_{\frac{1}{A.w\_clip}} + \lambda B\_normdev_{\frac{1}{B.w\_clip}} + \lambda C\_normdev_{\frac{1}{C.w\_clip}}} =$$

$$\frac{\lambda B\_normdev_{\frac{1}{A.w\_vew}}}{\lambda A\_normdev_{\frac{1}{A.w\_view}} + \lambda B\_normdev_{\frac{1}{B.w\_view}} + \lambda C\_normdev_{\frac{1}{C.w\_view}}} = \frac{\lambda B\_view}{1} = \lambda B\_view$$

$$\lambda C\_hw = \frac{\lambda C\_normdev_{\frac{1}{A.w\_clip}}}{\lambda A\_normdev_{\frac{1}{A.w\_clip}} + \lambda B\_normdev_{\frac{1}{B.w\_clip}} + \lambda C\_normdev_{\frac{1}{C.w\_clip}}} =$$

$$\frac{\lambda C\_normdev_{\frac{1}{A.w\_vew}}}{\lambda A\_normdev_{\frac{1}{A.w\_view}} + \lambda B\_normdev_{\frac{1}{B.w\_view}} + \lambda C\_normdev_{\frac{1}{C.w\_view}}} = \frac{\lambda C\_view}{1} = \lambda C\_view$$

双抛物面投影

双抛物面映射

Wolfgang Heidrich, Hans-Peter Seidel. "View-independent Environment Maps." EUROGRAPHICS 1998.
https://vccimaging.org/Publications/Heidrich1998VEM/Heidrich1998VEM.pdf

Imagination Technologies Limited. "Dual Paraboloid Environment Mapping." Power SDK Whitepaper 2017.
https://github.com/powervr-graphics/Native_SDK/tree/4.3/Documentation/Whitepapers

双抛物面（Dual Paraboloid）映射在本质上是一种投影函数（Projector Function），它将球函数（Spherical Function）的定义域（三维空间中的任意方向，可以等价地表示为三维空间中的任意单位向量，一般用于表示从三维空间中的某个固定点出发的各个方向，可以形象地理解为从球心出发到球面上的任意点的方向）映射到二维空间中的坐标，以允许用图像（Image）来保存球函数。

我们设（椭圆）抛物面$z = \frac{1}{2} - \frac{1}{2}(x^2 + y^2)$，不难理解它是由准线为 z = 1、顶点为$\left(0,\ 0,\ \frac{1}{2}\right)$、焦点为原点的抛物线绕 Z 轴旋转一周得到的曲面，简单的说明如下：

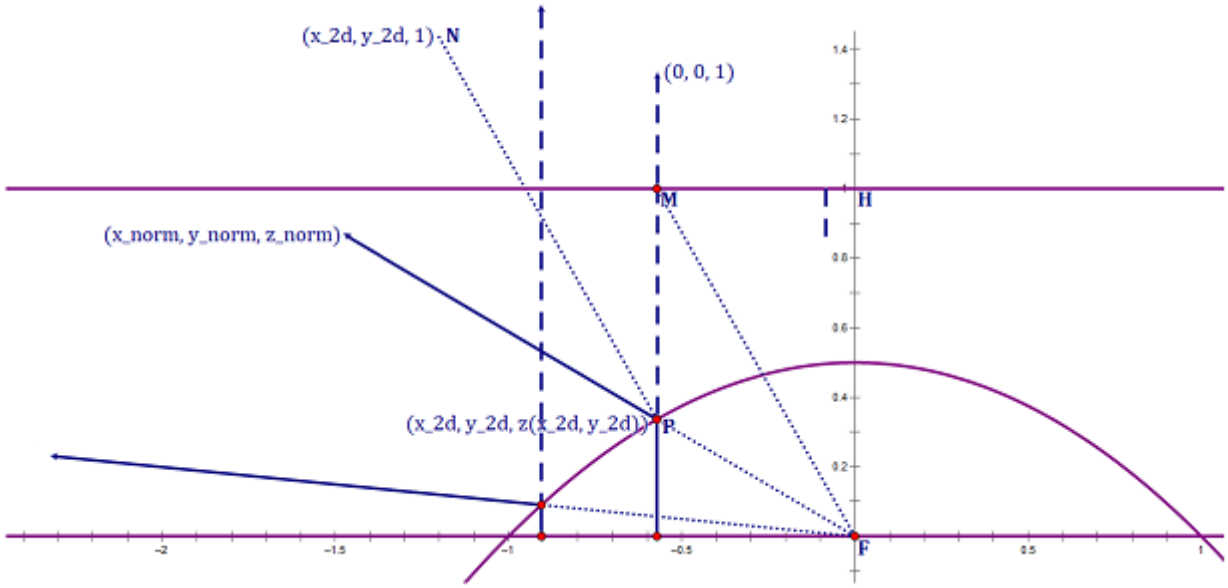
令 y = 0，我们得到抛物面 $z = \frac{1}{2} - \frac{1}{2}(x^2 + y^2)$与平面 ZOX 的交线$z = \frac{1}{2} - \frac{1}{2}x^2$，它是由抛物线$z = -\frac{1}{2}x^2$向 Z 轴正方向平移$\frac{1}{2}$个单位得到的。

而抛物线$z = -\frac{1}{2}x^2$又可以写成 x2 = -2z，对照标准形式 x2 = -2pz，可得 p = 1，即抛物线$z = -\frac{1}{2}x^2$的准线为$z = \frac{p}{2} = \frac{1}{2}$、顶点为$(0,\ 0)$、焦点为$\left(0,\ -\frac{p}{2}\right) = \left(0,\ -\frac{1}{2}\right)$。

将抛物线$z = -\frac{1}{2}x^2$向 Z 轴正方向平移$\frac{1}{2}$个单位后即得到抛物线$z = -\frac{1}{2}x^2$，它的准线为$z = \frac{1}{2} + \frac{1}{2} = 1$、顶点为

$\left(0,\quad 0+\frac{1}{2}\right)=\left(0,\quad \frac{1}{2}\right)$、焦点为$\left(0,\quad -\frac{1}{2}+\frac{1}{2}\right)=(0,\quad 0)$。

设球函数的定义域中的某个自变量为(x_norm, y_norm, z_norm)，双抛物面映射的目标是将它映射到的二维空间中的坐标(x_2d, y_2d)，下面会对该映射过程进行说明：



设从焦点 F 出发方向为(x_norm, y_norm, z_norm)的射线与抛物面交于点 P，过点 P 作准线的垂线段 PM，表示 PM 方向的单位向量为(0, 0, 1)。

与向量 FP 和向量 PM 的夹角的角平分线 PN 的方向相同的向量为(x_norm, y_norm, z_norm) + (0, 0, 1) = (x_norm, y_norm, z_norm + 1)。

在四维齐次空间内，以上过程可以用线程变换表示为$\begin{bmatrix} x_norm \\ y_norm \\ z_norm\ +\ 1 \\ 1 \end{bmatrix}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_norm \\ y_norm \\ z_norm \\ 1 \end{bmatrix}$。

根据抛物线的性质，|PM| = |PF|，ΔPMF 是等腰三角形，∠NPM = ∠FMP，PN // FM。

又因为$\overrightarrow{FM}=\overrightarrow{FH}+\overrightarrow{HM}=(0,\quad 0,\quad 1)+(x_2d,\quad x_2d,\quad 0)=(x_2d,\quad x_2d,\quad 1)$。
根据共线向量基本定理，存在唯一的实数 λ，使$\overrightarrow{PN}=\lambda\overrightarrow{FM}$，即(x_norm, y_norm, z_norm + 1) = λ(x_2d, y_2d, 1)，即$(x_2d\quad y_2d)=\frac{1}{z_norm+1}(x_norm\quad y_norm)$。

在四维齐次空间内，以上过程可以用线程变换表示为$\begin{bmatrix} x_2d \\ y_2d \\ * \\ 1 \end{bmatrix}=\frac{1}{z_norm+1}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & b & a \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_norm \\ y_norm \\ z_norm\ +\ 1 \\ 1 \end{bmatrix}$。//矩阵中的第 4 行模仿透视投影变换，将 z_norm + 1 写入 w 分量，在使用双抛物面映射生成图像时，数乘$\frac{1}{z_norm+1}$的操作由图形流水线中的固定功能阶段完成；矩阵中的 a 和 b 与透视投影变换中的相同，透视投影中的 Reverse-Z 等相关技术也可以应用到此处。

综上所述，在四维齐次空间内，整个双抛物面映射的过程可以用线性变换$\begin{bmatrix} x_2d \\ y_2d \\ * \\ 1 \end{bmatrix}=$

$\frac{1}{z\_norm+1}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&b&a\\0&0&1&0\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&1\\0&0&0&1\end{bmatrix}\begin{bmatrix}x\_norm\\y\_norm\\z\_norm\\1\end{bmatrix}$。 //其中，在使用双抛物面映射生成图像时，数乘$\frac{1}{z_norm+1}$的操作由图形流水线中的固定功能阶段完成

虽然可以根据以上过程可以将球函数的整个定义域映射到二维空间中，但是从焦点 F 出发方向接近 Z 轴负方向的射线，与抛物面的交点接近于无限远处，球函数的整个定义域在二维空间中对应的区域的面积并不是有限的。

因此，我们只将三维空间中 z 分量>0 的方向用抛物面$z = \frac{1}{2} - \frac{1}{2}(x^2+y^2)$映射到二维空间中；对于三维空间中 z 分量<0 的方向，我们用另一个抛物面$z = -\frac{1}{2} + \frac{1}{2}(x^2+y^2)$进行映射，从而使球函数的整个定义域映射到二维空间中的面积有限的区域（不难证明该区域的边界是一个以原点为中心、半径为 1 的圆）。

但是这会使我们得到 2 个图像，分别对应于三维空间中 z 分量>0 和 z 分量<0 的方向。传统的实现（比如 Heidrich 在 1998 年给出的实现）用纹理地图集（Texture Atlas）来保存这 2 个图像。由于传统的纹理地图集具有一定的缺陷：比如在生成 MipMap 时，这 2 个图像的交界处的融合可能会导致走样（Aliasing），现代的实现更倾向于用纹理数组（Texture Array）来保存这 2 个图像。在接下来的讨论中，在没有特殊说明的情况下，我们会假定实现使用现代的纹理数组而非传统的纹理地图集。

用双抛物面映射读取图像

首先我们用双抛物面映射将三维空间中的方向(x_norm, y_norm, z_norm)（比如：环境映射中的反射方向（根据视线方向和表面法线计算得到）、泛光灯阴影映射中的光线方向）映射到二维空间中的(x_2d, y_2d)。随后我们需要用一个线性变换将(x_2d, y_2d)映射到纹理坐标 UV，并根据三维空间中的方向的 z 分量>0 或 z 分量<0 确定纹理数组的索引。

用双抛物面映射生成图像

在某些情况下，我们需要用双抛物面映射生成图像来保存球函数（比如：环境映射用球函数表示光探头在三维空间中各个方向上受到的光照，泛光灯阴影映射用球函数表示泛光灯在三维空间中各个方向上到最近的几何体的距离）。

我们设三维空间中几何体的顶点在 ModelSpace 中的坐标为(x_model, y_model, z_model)，与绘制普通的几何体时一样，我们用 M 和 V 矩阵将它变换到 ViewSpace，即$\begin{bmatrix}x_view\\y_view\\z_view\\1\end{bmatrix}$= VM$\begin{bmatrix}x_model\\y_model\\z_model\\1\end{bmatrix}$。

我们将从原点出发到(x_view, y_view, z_view)的方向作为球函数的定义域中的自变量，即(x_norm, y_norm, z_norm) = normalize(x_view, y_view, z_view)；用双抛物面映射将它映射到二维空间中的(x_2d, y_2d)。由于(x_2d, y_2d)的 x 和 y 分量的范围在$[-1, 1]$之间，恰好与 NormalizedDeviceSpace 一致，不再需要进行其它变换。

双抛物面投影的透视校正插值

本人原创

我们回忆用双抛物面映射生成图像的过程（为了行文简洁，我们只描述了其中一个抛物面，显然这并不会影响到我们的讨论）：

ModelSpace->WorldSpace->ViewSpace

$$\begin{bmatrix} x\_view \\ y\_view \\ z\_view \\ 1 \end{bmatrix} = VM \begin{bmatrix} x\_model \\ y\_model \\ z\_model \\ 1 \end{bmatrix}$$

ViewSpace->SphericalFunctionDomain

$$(x\_viewnorm,\ \ y\_viewnorm,\ \ z\_viewnorm) = \mathrm{normalize}(x\_view,\ \ y\_view,\ \ z\_view)$$

SphericalFunctionDomain->ClipSpace

$$\begin{bmatrix} x\_clip \\ y\_clip \\ z\_clip \\ w\_clip \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & b & a \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x\_viewnorm \\ y\_viewnorm \\ z\_viewnorm \\ 1 \end{bmatrix}$$ //w_clip = z_viewnorm + 1

ClipSpace -> NormalizedDeviceSpace

$$\begin{bmatrix} x\_normdev \\ y\_normdev \\ z\_normdev \\ 1 \end{bmatrix} = \begin{bmatrix} x\_2d \\ y\_2d \\ * \\ 1 \end{bmatrix} = \frac{1}{w\_clip} \begin{bmatrix} x\_clip \\ y\_clip \\ z\_clip \\ w\_clip \end{bmatrix}$$ //由图形流水线中的固定功能阶段完成

我们设图形流水线中的光栅化阶段按照 λ_hw 进行插值。

在透视投影中：我们先用线性变换将向量从 ModelSpace 变换到 ClipSpace：$\begin{bmatrix} x\_clip \\ y\_clip \\ z\_clip \\ w\_clip \end{bmatrix} = PVM \begin{bmatrix} x\_model \\ y\_model \\ z\_model \\ 1 \end{bmatrix}$；随后用数乘（由图形流水线中的固定功能阶段完成）将向量从 ClipSpace 变换到 NormalizedDeviceSpace：$\begin{bmatrix} x\_normdev \\ y\_normdev \\ z\_normdev \\ 1 \end{bmatrix} = \frac{1}{w\_clip} \begin{bmatrix} x\_clip \\ y\_clip \\ z\_clip \\ w\_clip \end{bmatrix}$。

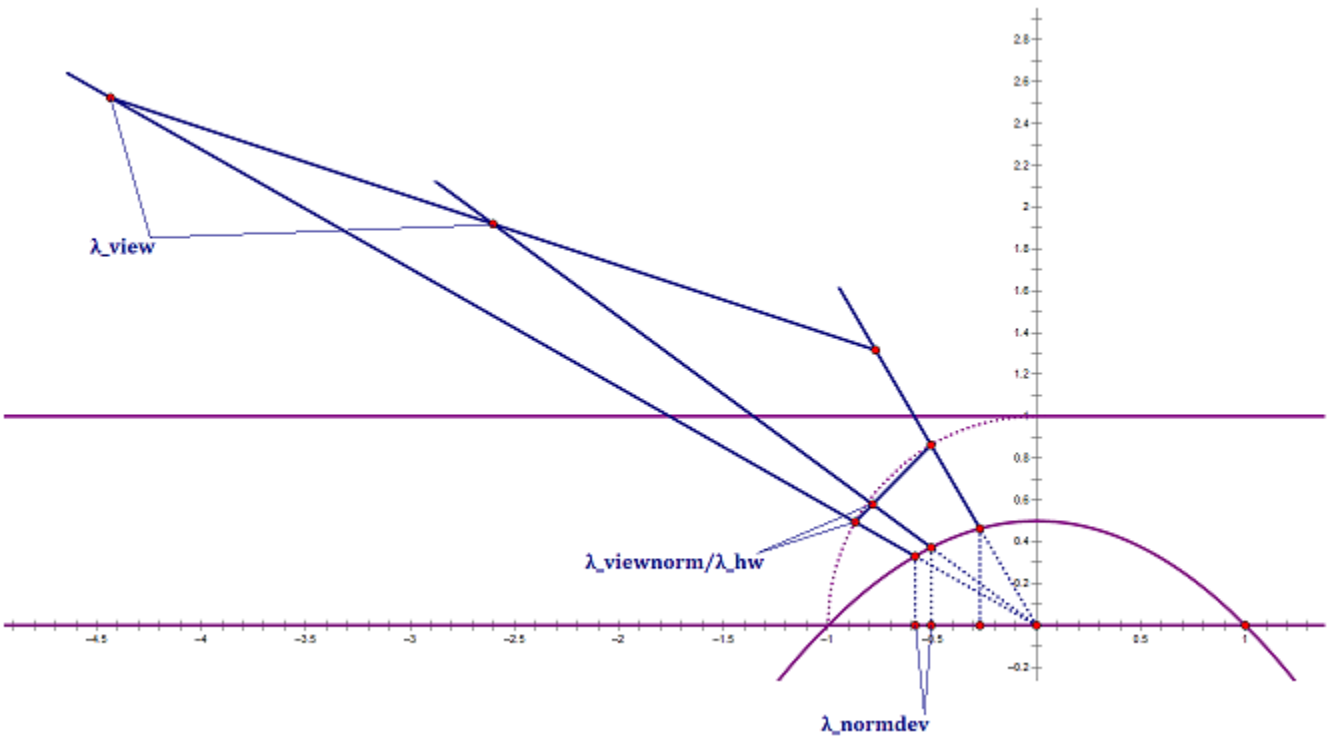
根据透视投影的透视校正插值中的相关讨论，我们有 λ_hw = λ_model。

在双抛物面映射中：我们先用线性变换将向量从 SphericalFunctionDomain 变换到 ClipSpace：$\begin{bmatrix} x\_clip \\ y\_clip \\ z\_clip \\ w\_clip \end{bmatrix} =$

$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&b&a\\0&0&1&0\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&1\\0&0&0&1\end{bmatrix}\begin{bmatrix}x\_viewnorm\\y\_viewnorm\\z\_viewnorm\\1\end{bmatrix}$；随后用数乘（由图形流水线中的固定功能阶段完成）将向量从 ClipSpace 变换到 NormalizedDeviceSpace：$\begin{bmatrix}x\_normdev\\y\_normdev\\z\_normdev\\1\end{bmatrix}=\frac{1}{w\_clip}\begin{bmatrix}x\_clip\\y\_clip\\z\_clip\\w\_clip\end{bmatrix}$。

对比透视投影的过程，我们不难理解有 λ_hw = λ_viewnorm 成立（详细的证明还是应当参考 Blinn 在 1992 年在论文双曲插值中的相关讨论）。

注：使用传统的纹理地图集的实现在 ClipSpace -> NormalizedDeviceSpace 中需要再乘上某个矩阵，将(x_2d, y_2d)映射到 NormDeviceSpace 的一半（上下分或左右分，具体的计算方式取决于的实现），即$\begin{bmatrix}x_normdev\\y_normdev\\z_normdev\\1\end{bmatrix}=M_H\begin{bmatrix}x_2d\\y_2d*\\1\end{bmatrix}=M_H\frac{1}{w_clip}\begin{bmatrix}x_clip\\y_clip\\z_clip\\w_clip\end{bmatrix}$。由于数乘满足交换律，我们有$\begin{bmatrix}x_normdev\\y_normdev\\z_normdev\\1\end{bmatrix}=M_H\frac{1}{w_clip}\begin{bmatrix}x_clip\\y_clip\\z_clip\\w_clip\end{bmatrix}=\frac{1}{w_clip}M_H\begin{bmatrix}x_clip\\y_clip\\z_clip\\w_clip\end{bmatrix}$，整个过程仍可以看作是一个线性变换$\begin{bmatrix}x_clip_H\\y_clip_H\\z_clip_H\\w_clip_H\end{bmatrix}=M_H\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&b&a\\0&0&1&0\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&1\\0&0&0&1\end{bmatrix}\begin{bmatrix}x_viewnorm\\y_viewnorm\\z_viewnorm\\1\end{bmatrix}$和一个数乘$\begin{bmatrix}x_normdev\\y_normdev\\z_normdev\\1\end{bmatrix}=\frac{1}{w_clip_H}\begin{bmatrix}x_clip_H\\y_clip_H\\z_clip_H\\w_clip_H\end{bmatrix}$组成。



由于 ModelSpace->WorldSpace->ViewSpace 是线性变换，我们有 λ_view = λ_world = λ_model。但是

ViewSpace->SphericalFunctionDomain 并不是线性变换，因此 λ_viewnorm ≠ λ_view，从而导致图形流水线中的光栅化阶段所使用的 λ_hw 不再与三维空间中的 λ_view 相等。

我们只要找到某种方式，基于 λ_viewnorm 求出 λ_view，就可以做到按照三维空间中的比例 λ_model 对顶点属性 VTXATTR 进行插值。

我们设三维空间中共面的四点 P、A、B、C 在 ViewSpace 中的坐标为$\overrightarrow{OP_{View}}$、$\overrightarrow{OA_{View}}$、$\overrightarrow{OB_{View}}$、$\overrightarrow{OC_{View}}$；设$k\overrightarrow{OP_{View}}$（$k \neq 0$）为 OP_{View} 与$\triangle A_{ViewNorm}B_{ViewNorm}C_{ViewNorm}$ 的交点的坐标。

由于 λ_hw = λ_viewnorm，我们有$\overrightarrow{OP_{View}} = \lambda A_hw\frac{\overrightarrow{OA_{View}}}{|\overrightarrow{OA_{View}}|} + \lambda B_hw\frac{\overrightarrow{OB_{View}}}{|\overrightarrow{OB_{View}}|} + \lambda C_hw\frac{\overrightarrow{OC_{View}}}{|\overrightarrow{OC_{View}}|}$，等式两边同时除以 k 即

$\overrightarrow{OP_{View}} = (\frac{1}{k}\lambda A\_hw\frac{1}{|\overrightarrow{OA_{View}}|})\overrightarrow{OA_{View}} + (\frac{1}{k}\lambda B\_hw\frac{1}{|\overrightarrow{OB_{View}}|})\overrightarrow{OB_{View}} + (\frac{1}{k}\lambda C\_hw\frac{1}{|\overrightarrow{OC_{View}}|})\overrightarrow{OC_{View}}$。（等式Ⅰ）

由于 P、A、B、C 四点共面，根据共面向量定理，我们有：
存在唯一的实数 λA_view、λB_view、λC_view，使$\overrightarrow{OP_{View}} = \lambda A_view\overrightarrow{OA_{View}} + \lambda B_view\overrightarrow{OB_{View}} + \lambda C_view\overrightarrow{OC_{View}}$，且 λA_view + λB_view + λC_view = 1；

由于唯一性，我们有：

$\lambda A\_view = \frac{1}{k}\lambda A\_hw\frac{1}{|\overrightarrow{OA_{View}}|}$ （等式Ⅱ）

$\lambda B\_view = \frac{1}{k}\lambda B\_hw\frac{1}{|\overrightarrow{OB_{View}}|}$ （等式Ⅲ）

$\lambda C\_view = \frac{1}{k}\lambda C\_hw\frac{1}{|\overrightarrow{OC_{View}}|}$ （等式Ⅳ）

$\frac{1}{k}\lambda A\_hw\frac{1}{|\overrightarrow{OA_{View}}|} + \frac{1}{k}\lambda B\_hw\frac{1}{|\overrightarrow{OB_{View}}|} + \frac{1}{k}\lambda C\_hw\frac{1}{|\overrightarrow{OC_{View}}|} = 1$，等式两边同时乘 k 即$k = \lambda A_hw\frac{1}{|\overrightarrow{OA_{View}}|} + \lambda B_hw\frac{1}{|\overrightarrow{OB_{View}}|} +$

$\lambda C\_hw\frac{1}{|\overrightarrow{OC_{View}}|}$ （等式Ⅴ）

观察等式Ⅰ的形式，我们尝试将$VTXATTRORIGIN * \frac{1}{\text{Length}(\begin{bmatrix}x_view\\y_view\\z_view\end{bmatrix})}$ 作为 VTXATTR 输出到 Rasterization，我们将在 PixelShader 中得到 VTXATTR 输入：$P.VtxAttrOrigin\_Div\_LengthPositionView = \lambda A\_hw * A.VTXATTRORIGIN * \frac{1}{\text{Length}(\begin{bmatrix}A.x\_view\\A.y\_view\\A.z\_view\end{bmatrix})} + \lambda B\_hw * B.VTXATTRORIGIN * \frac{1}{\text{Length}(\begin{bmatrix}B.x\_view\\B.y\_view\\B.z\_view\end{bmatrix})} + \lambda C\_hw * C.VTXATTRORIGIN *$

$\frac{1}{\text{Length}(\begin{bmatrix}C.x\_view\\C.y\_view\\C.z\_view\end{bmatrix})} = \lambda A\_hw\frac{1}{|\overrightarrow{OA_{View}}|}A.VTXATTRORIGIN + \lambda B\_hw\frac{1}{|\overrightarrow{OB_{View}}|}B.VTXATTRORIGIN +$

$\lambda C\_hw\frac{1}{|\overrightarrow{OC_{View}}|}C.VTXATTRORIGIN$。

观察得到的等式的形式，我们发现与等式Ⅰ恰好相差$\frac{1}{k}$，我们尝试将$\frac{1}{\text{Length}(\begin{bmatrix}x_view\\y_view\\z_view\end{bmatrix})}$作为 VTXATTR 输出到 Rasterization，

我们将在 PixelShader 中得到 VTXATTR 输入：$P.One\_Div\_LengthPositionView = \lambda A\_hw * \frac{1}{\text{Length}(\begin{bmatrix}A.x\_view\\A.y\_view\\A.z\_view\end{bmatrix})} + \lambda B\_hw *$

$$\frac{1}{\mathrm{Length}\left(\begin{bmatrix}\mathrm{B.x\_view}\\ \mathrm{B.y\_view}\\ \mathrm{B.z\_view}\end{bmatrix}\right)} + \lambda\mathrm{C\_hw} * \frac{1}{\mathrm{Length}\left(\begin{bmatrix}\mathrm{C.x\_view}\\ \mathrm{C.y\_view}\\ \mathrm{C.z\_view}\end{bmatrix}\right)} = \lambda\mathrm{A\_hw}\frac{1}{|\overrightarrow{\mathrm{OA_{View}}}|} + \lambda\mathrm{B\_hw}\frac{1}{|\overrightarrow{\mathrm{OB_{View}}}|} + \lambda\mathrm{C\_hw}\frac{1}{|\overrightarrow{\mathrm{OC_{View}}}|} = \mathrm{k}$$（根据等式Ⅴ），恰好为 k。

我们将以上两个结果相除即可得到按照三维空间中的比例 λ_model 对顶点属性 VTXATTRORIGIN 进行插值的结果：

$$\frac{\mathrm{P.VtxAttrOrigin\_Div\_LengthPositionView}}{\mathrm{P.One\_Div\_LengthPositionView}} = \frac{1}{\mathrm{k}}\lambda\mathrm{A\_hw}\frac{1}{|\overrightarrow{\mathrm{OA_{View}}}|}\mathrm{A.VTXATTRORIGIN} + \frac{1}{\mathrm{k}}\lambda\mathrm{B\_hw}\frac{1}{|\overrightarrow{\mathrm{OB_{View}}}|}\mathrm{B.VTXATTRORIGIN} + \frac{1}{\mathrm{k}}\lambda\mathrm{C\_hw}\frac{1}{|\overrightarrow{\mathrm{OC_{View}}}|}\mathrm{C.VTXATTRORIGIN} = \lambda\mathrm{A\_view} * \mathrm{A.VTXATTRORIGIN} + \lambda\mathrm{B\_view} * \mathrm{B.VTXATTRORIGIN} + \lambda\mathrm{C\_view} * \mathrm{C.VTXATTRORIGIN}$$（根据等式Ⅱ、Ⅲ、Ⅳ）

综上所述，在双抛物面映射中正确进行透视校正插值的方法为：

1.将正确的 w_clip 值（比如 z_viewnorm + 1）写入到 gl_position (Vulkan)/ SV_Position0(D3D12)的 w 分量。

2.对任意需要正确透视校正插值的 VTXATTRORIGIN，用 VertexShader/TessellationEvaluationShader/GeometryShade 输出 VTXATTRORIGIN×$\frac{1}{\mathrm{Length}\left(\begin{bmatrix}\mathrm{x_{view}}\\ \mathrm{y_{view}}\\ \mathrm{z_{view}}\end{bmatrix}\right)}$到 Rasterization，Rasterization 双曲插值后在 PixelShader 中对应的属性记作 VtxAttrOrigin_Div_LengthPositonView。

3.用 VertexShader/TessellationEvaluationShader/GeometryShader 添加顶点属性$\frac{1}{\mathrm{Length}\left(\begin{bmatrix}\mathrm{x_{view}}\\ \mathrm{y_{view}}\\ \mathrm{z_{view}}\end{bmatrix}\right)}$并输出到 Rasterization，Rasterization 双曲插值后在 PixelShader 中对应的属性记作 One_Div_LengthPositionView。

4.在 PixelShader 中将 VtxAttrOrigin_Div_LengthPositonView 和 One_Div_LengthPositionView 相除即可得到正确透视校正插值的 VtxAttrOrigin。