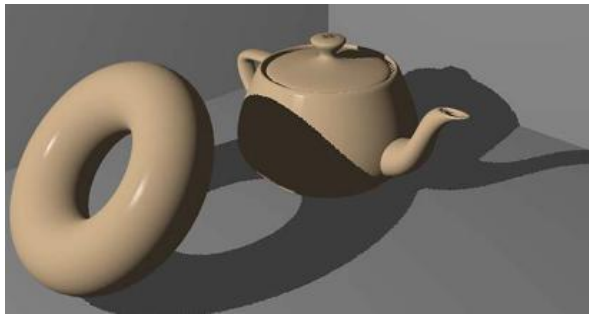
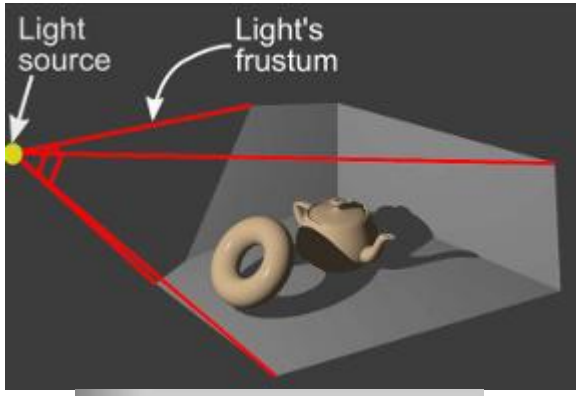
Ombres

Malek.Bengougam@gmail.com

Projective Shadow Mapping (rappels)



- Rendu hors écran depuis le point de vue de la lumière



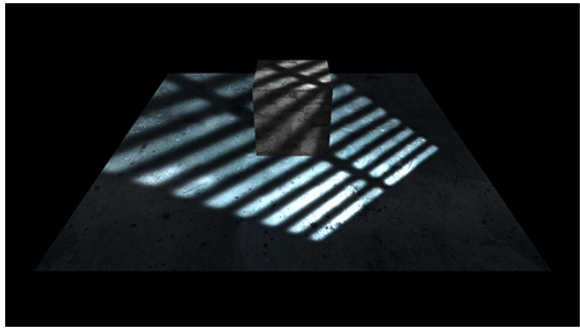
- Une LookAt (orientation) et une matrice de projection (frustum)
  - Rappel: LookAt est une matrice inverse (world -> local space)
  - Projection: orthographique (directional) ou perspective (omni, spot)
    - Omnidirectional : 6 points de vue avec un FOV de 90°

- Render Target carrée ok et puissance de 2 ok
  - Généralement depth seulement

- Fonctionnement similaire en forward/deferred
  - Deferred: shadowing dans la passe d'illumination

Projected (drop) shadows

- L'aspect projectif consiste à projeter (plaquer) la texture.
  - On peut utiliser une texture pré-générée
  - Dans le cas du shadow mapping, la texture est procedurale (render target)



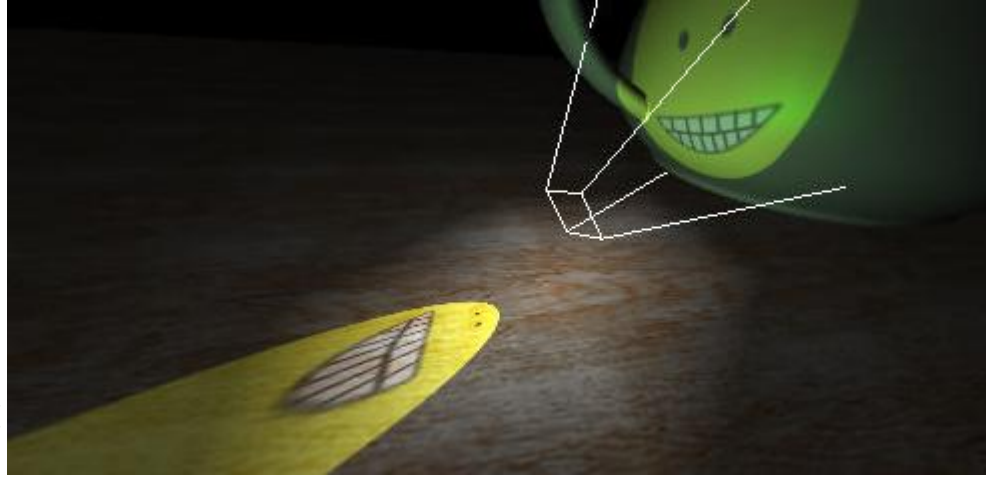
- C'est le receveur (receiver) qui échantillonne la texture projetée par l'émetteur (caster/blocker).

- On peut utiliser cette technique également avec une texture standard pour former un décalque (posée = drop).
  - Dans ce cas on a seulement besoin des matrices du projecteur

https://stackoverflow.com/questions/22732717/opengl-projective-texture-mapping-via-shaders

https://simonschreibt.de/gat/infinity-nikki-shadow/

Projective Textures robustes



- Il faut tout de même faire attention à certains points :

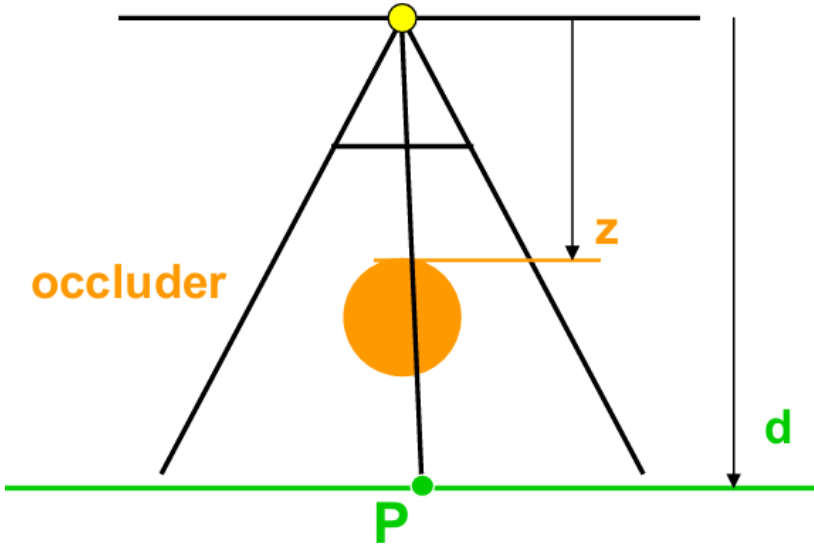
- Bien que l'on ramène les coordonnées à [-1;1] en divisant .xyz par .w (ce qui normalise en quelque sorte)

- On peut quand même avoir des valeurs en dehors de ce domaine
  - Il faut donc clamper ici … et définir une couleur "vide" pour les texels en dehors (blanc, noir…la fameuse "border color" de glTexImage2D).

- W peut lui aussi être négatif ! Ce qui produit une image inversée.
  - Là encore il faut clamper pour éviter une projection 'reverse'.

https://developer.download.nvidia.com/assets/gamedev/docs/projective_texture_mapping.pdf

Shadow mapping



- On compare les distances en light space

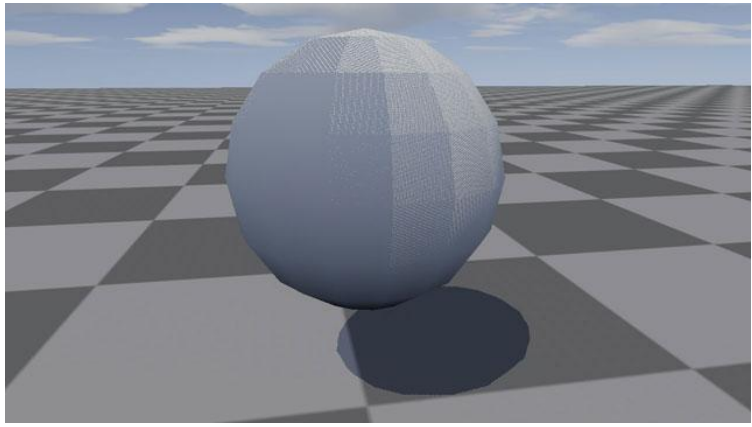
- Se rappeler que la shadow map contient un Z (depth value) exprimé dans le repère de la source lumineuse.
  - Pour comparer le 'Z' du pixel il faut être dans le même repère que $Z_{shadowmap}$
  - On multiplie $xyzw_{pixel}$ par $Light_{view}$ et $Light_{proj}$

```
vec4 Plight = LightProj * LightView * Pworld;
Plight.xyz = Plight.xyz * 0.5 + 0.5; // (0;+1), peut etre fait apres la division ou integre a la matrice de projection
Plight.xyz = Plight.xyz / Plight.w; // division perspective, pas necessaire en Ortho (-1;+1)
vec2 shadowCoords = Plight.xy;
if (texture(shadowMap, shadowCoords).z < Plight.z) { // in shadow }
```

- alternativement avec textureProj()

```
Plight.xyz = Plight.xyz * 0.5 + 0.5;
vec3 shadowCoords = Plight.xyw;
if (textureProj(shadowmap, shadowCoords).z < Plight.z) { // in shadow }
```

Problème #1 : Acné de surface (1)



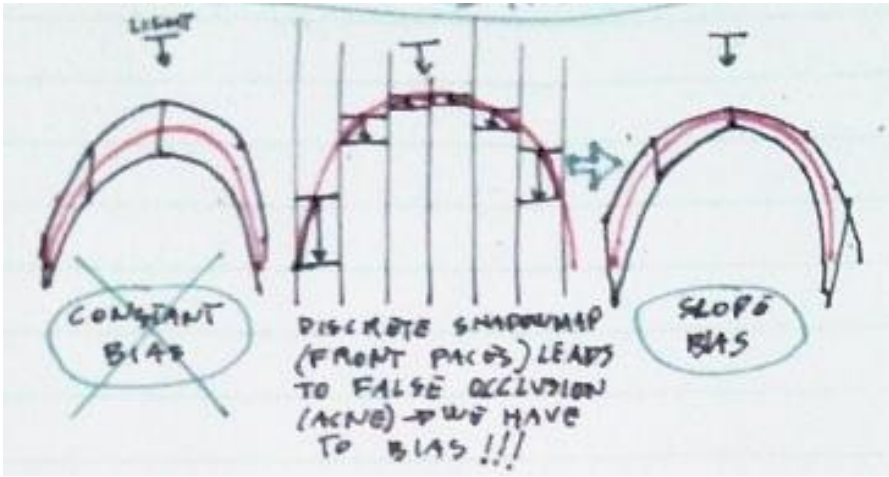
- Problème qui ne peut pas être fixé par une améliration du sampling ou de la résolution de la texture

- Supprimer les faces avants (front) peut aider parfois…mais risque d'introduire des artefacts également (light leaks).
  - Vrai en particulier pour les objets et murs fins.
  - Peut donc se fixer en ajoutant une "épaisseur" à la géométrie.

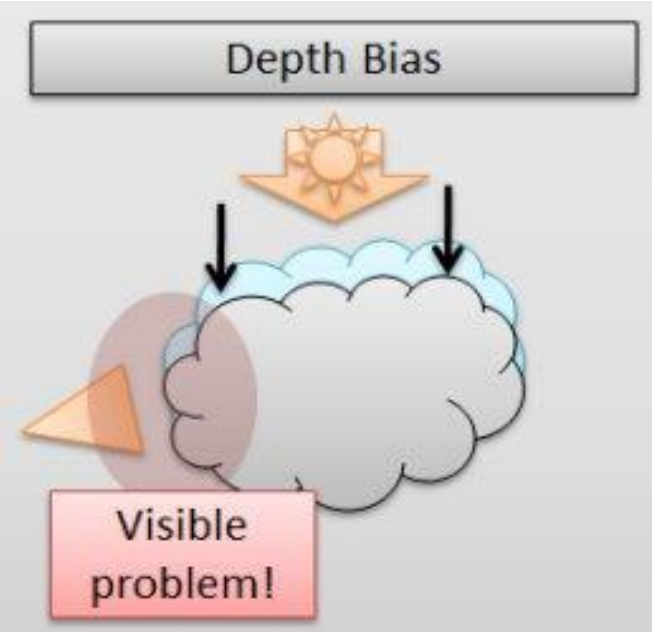
- Shadow bias : les distances de la shadow map doivent être rapprochés de la lumière.
  - Unité: la valeur du biais est exprimée en texels de la shadow map (dépend de la résolution).
  - Exemple: **float** bias **=** texelSize ***** **tan**(acos(NdotL));

- Support GPU: slope-scale depth bias. Le GPU prend en compte automatiquement la pente de la surface du caster (au moment de l'écriture/sampling du depth).

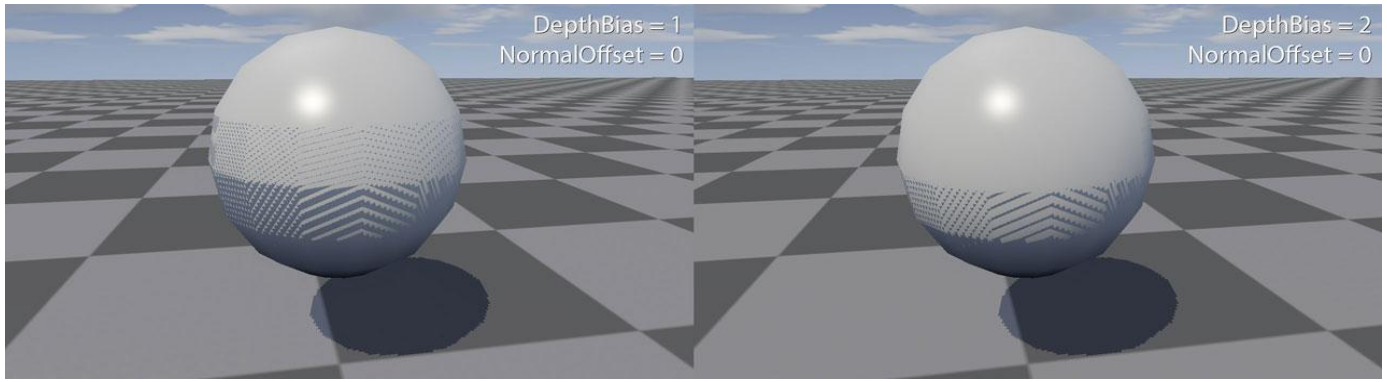
  - A défaut on intègre le bias dans le test : `Plight.z -= bias;`



Problème #1 : Limites du biais

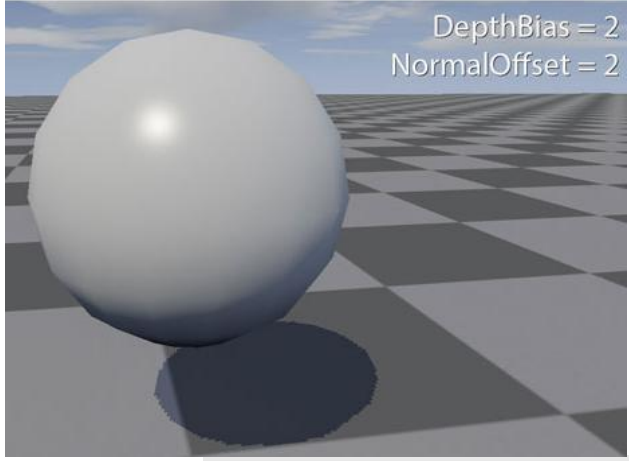


- Pas suffisant, les problèmes d'acné persistent



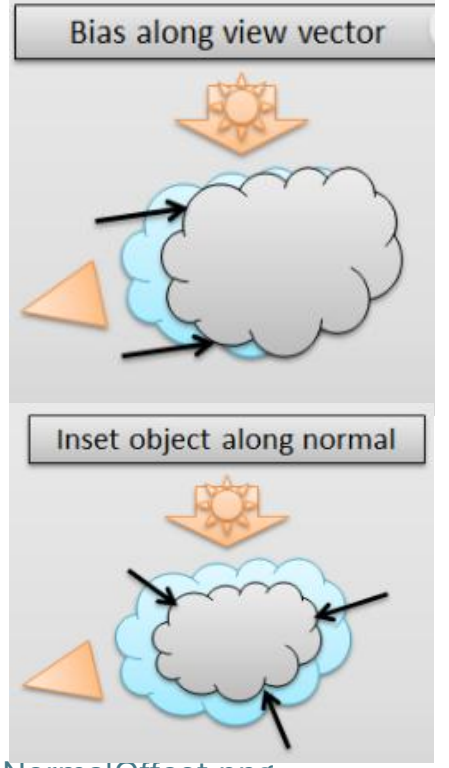
- Le biais est dépendant de la projection: plus faible proche de la caméra, et plus grand pour les texels éloignés
- Le biais dépend également du filtrage des ombres. Plus le filtre est large, plus le biais doit être important.
- Un biais trop important introduit un détachement non naturel de l'ombre et du blocker : "Peter-panning"

Fixer les problèmes d'Acné

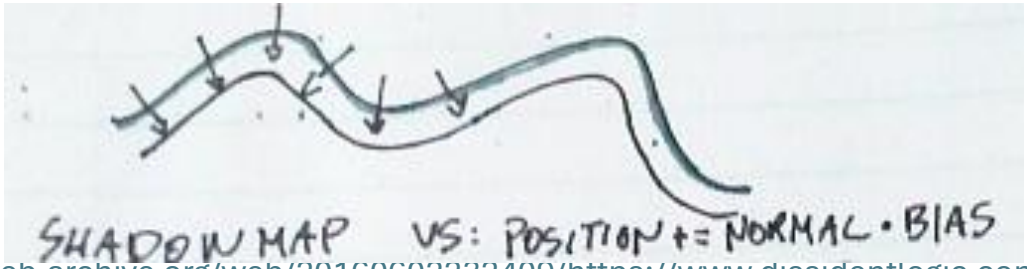


- Les techniques basées sur un biais ne sont pas suffisantes. Alternativement :

- On peut appliquer un biais supplémentaire en direction de la caméra
  - On ne fait que déplacer le problème de light space vers camera space
  - Peut être acceptable selon l'orientation respective de la caméra et des lumières



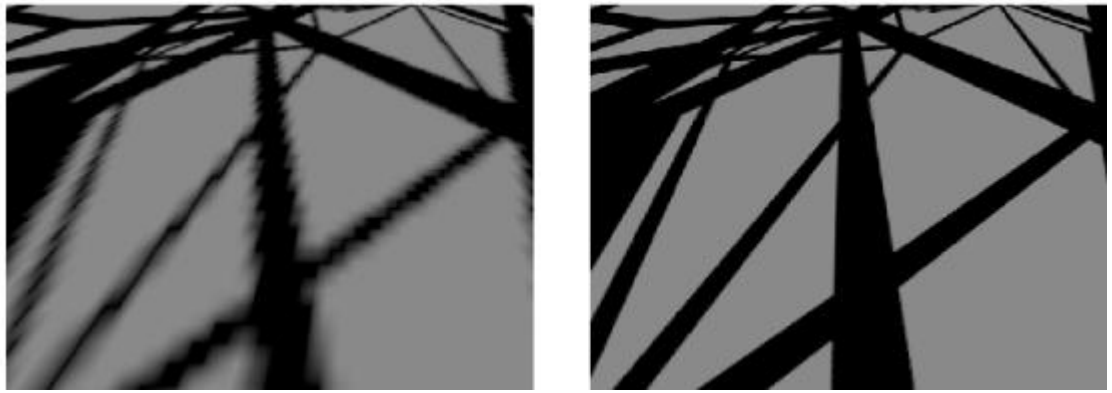
- Une meilleure méthode consiste à décaler en suivant la direction de la normale



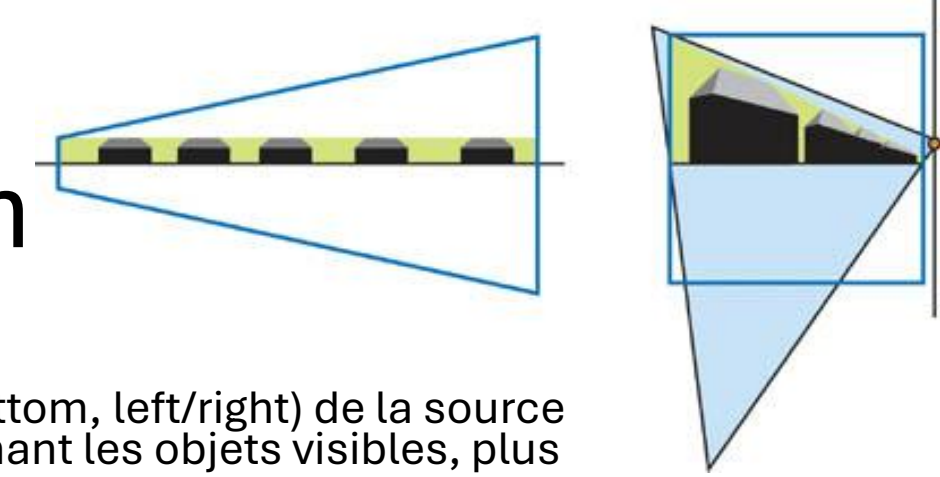
https://web.archive.org/web/20160602232409/https://www.dissidentlogic.com/old/images/NormalOffsetShadows/GDC_Poster_NormalOffset.png

Problème #2: distribution des texels

- Le principal facteur qui influe sur la qualité des ombres reste la distribution des texels
- Augmenter la résolution permet d'améliorer le nombre de sous-pixels, cependant on perd de grandes zones inutilisées.



Fixer le problème de résolution



- Cube clipping : on va réduire la zone de projection (near/far, top/bottom, left/right) de la source lumineuse vers la depth texture à un cube/parallélogramme contenant les objets visibles, plus spécifiquement les "receivers' seulement.

https://learn.microsoft.com/en-us/windows/win32/dxtecharts/common-techniques-to-improve-shadow-depth-maps#calculating-a-tight-projection

- Les plans near et far doivent par contre contenir l'ensemble des 'casters' idéalement.

https://learn.microsoft.com/en-us/windows/win32/dxtecharts/common-techniques-to-improve-shadow-depth-maps#calculating-a-tight-projection

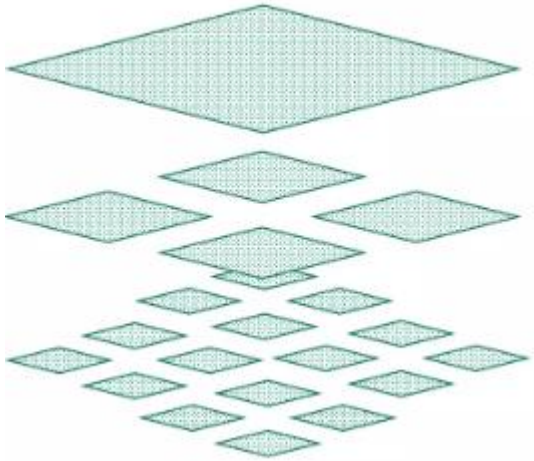
- Ultimement, dans le shader, vous pouvez clamper les positions projetées en light space avec les valeurs de near et far. C'est ce qu'on appelle le "pancaking".
  - Cela permet aussi de réduire la distance maximum stockée dans la depth map, et donc d'utiliser un format de donnée plus faible (D16 ou 16F).

Problème #3 : edge shimmering

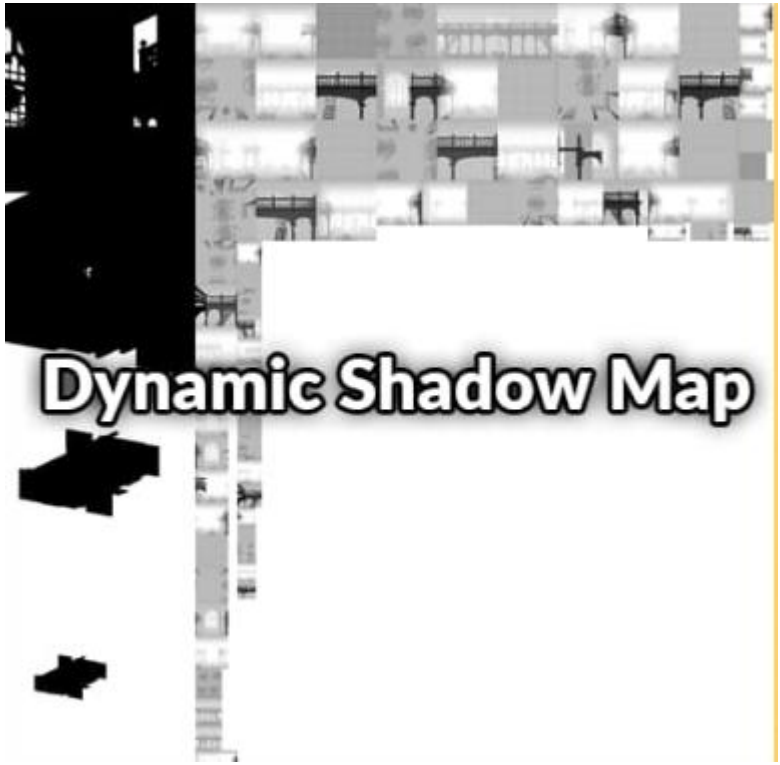
- C'est une forme d'aliasing / fighting : les texels des arêtes de l'ombre passe d'un pixel à un autre lorsque la caméra se déplace latéralement par ex.
  - Cela donne l'impression que les ombres flottent à la surface

- On peut fixer ce problème simplement en "snappant", càd en arrondissant les valeurs min/max de la boite orthographique à des valeurs entières ou multiples du ratio rangeMax / textureSize.

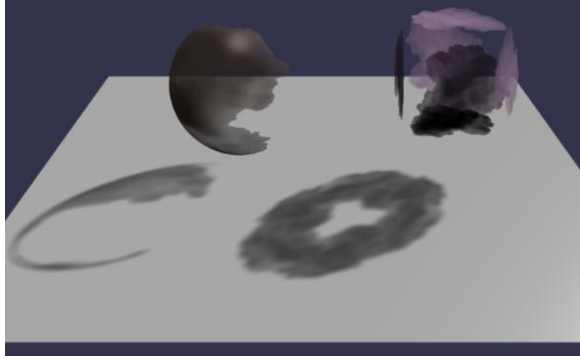
Caching



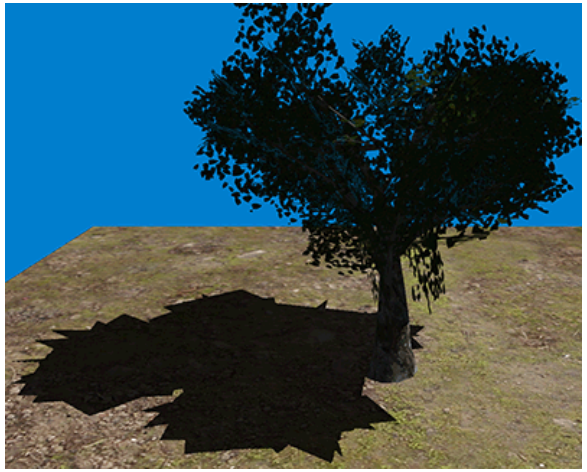
- Rendu séparé pour un ou plusieurs objets
  - On ne recalcule alors que très rarement les ombres pour ces objets
- Stockage en mémoire (une texture spécifique)
  - Souvent on utilise des atlas (regroupement de plusieurs petites textures)
- Cependant si les objets et/ou la lumière se déplacent il faut mettre à jour
  - Pas nécessaire de tout update d'un coup, étalez sur plusieurs trames
- Technique parfois nommée deferred shadows
  - à ne pas confondre avec l'utilisation des ombres en deferred rendering



Cas des objets (semi) transparents

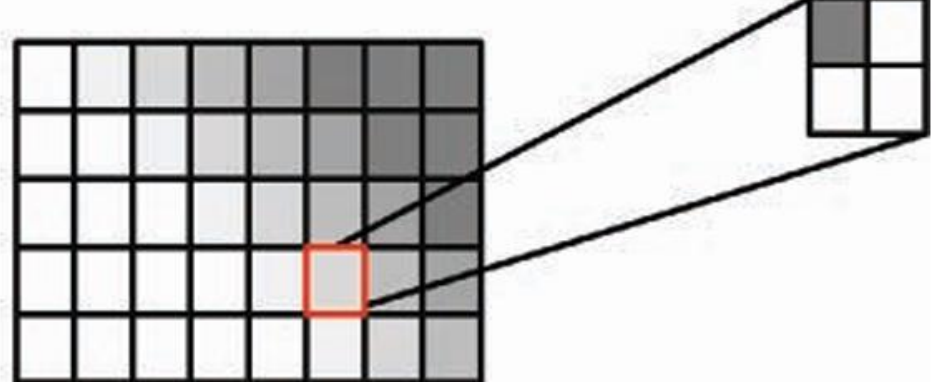


- Lorsque les objets sont partiellement transparents (comme les feuilles d'un arbre ou un logo en texture) il faut tenir compte de l'alpha dans la depth map.



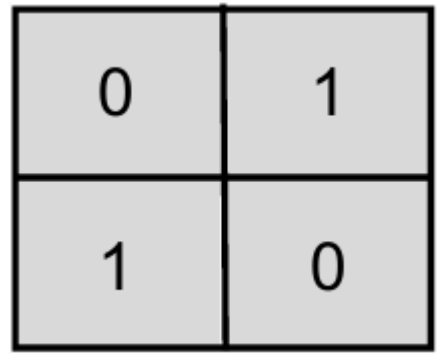
- On ajoute tout simplement un canal supplémentaire (idéalement dans une passe séparée) pour stocker l'opacité.

Filtrage des shadow maps (1)



- Une shadow map stocke des valeurs de profondeurs
- Le filtrage bilinéaire classique de répond pas aux critères ici : flouter les valeurs de profondeur introduit des erreurs dans la comparaison des distance (moyennes).

- Solution : PCF ou Percentage-Closer Filtering
- On effectue une moyenne non pas de la profondeur mais de la visibilité des texels.
  - On ne fait pas seulement une moyenne de 0 et de 1 mais on tient compte du différentiel d'UVs aussi



- Les GPUs supportent un PCF bilinéaire (2x2) quasiment gratuit
- Typiquement on utilise des kernels plus large (8x8, 9x9 etc...)
  - On peut réduire le nombre d'échantillons en utilisant des coordonnées entre deux texels.

- Visuellement, le PCF permet d'adoucir le rendu
  - Première façon basique de simuler de la pénombre (soft shadows) en lissant les bords.

Limites du PCF

FIGURE 5 Erroneous self-shadowing.

- Deux texels adjacents dans la depth map (dans le repère de la lumière) ne le sont pas forcément une fois projeté dans le repère NDC de la caméra.
  - De plus, il faut idéalement appliquer la transformation ViewSpace->LightSpace pour tous les échantillons, hors le Hardware n'a pas cette information !

- La valeur de biais est encore plus problématique ici: si un texel a une distance plus grande, le biais doit être plus large.....et donc un fort risque de Peter-panning.

- Une solution : calculer ce biais à l'aide des "derivatives" (dFdx/dFdy en GLSL, ddx/ddy en HLSL).
  Il faut pour calculer une matrice inverse des composants .xy (une mat 2x2) des derivatives (cf. le Jacobien d'une matrice).
- Permet de calculer un ratio en .xy que l'on applique aux .z des derivatives.
- On forme ensuite 2 vecteurs 2D (right et up) composés de (1.0/texelSize) * ratio, ce qui nous donne les directions (et pondération) pour notre filtre.
- https://ubm-twvideo01.s3.amazonaws.com/o1/vault/GD_Mag_Archives/GDM_May_2010.pdf pp 39-40

Filtrage des shadow maps (2)

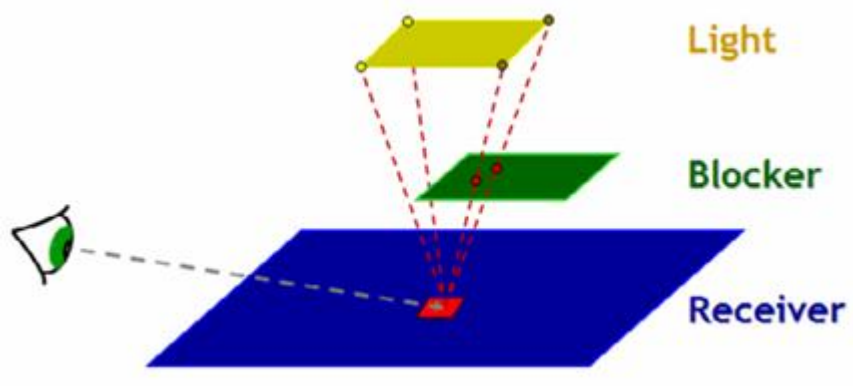
- On ajoute parfois une metadata supplémentaire (un canal en plus) pour faciliter le fitrage:
- ESM: exponential shadow map
- VSM: variance shadow map
- …ou EVSM, un mix des deux.
- MSM: Moment shadow map, pousse la logique du VSM à l'utilisation de 4 moments. Exemple : https://a-raven.github.io/PersonalWebpageCN/2019/10/07/4CS562Project2/

- Voir le readme.md ici pour les liens et plus d'infos

https://github.com/TheRealMJP/Shadows

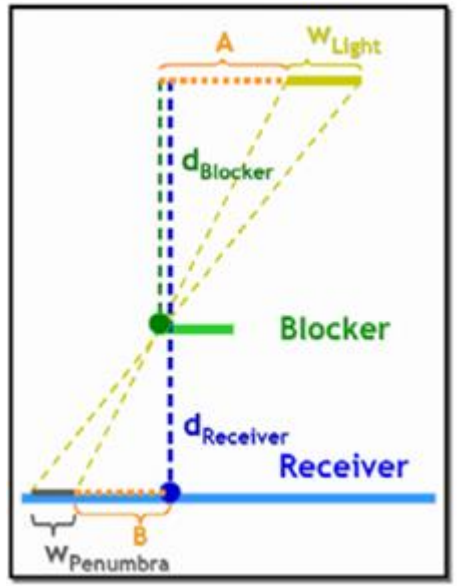
Contact harderning Shadows

- Un groupe de techniques qui améliore la qualité visuelle des ombres



- PCSS - Percentage-Closer Soft Shadows
- Observation initiale est que plus le kernel est large, plus l'ombre est douce.

- Etape 1 : recherche des blockers. Pour un pixel "receiver" on recherche les texels adjacents dans la shadow map qui sont plus proches de la lumière (rectangle rouge ici, dépend de la distance et de la source).
- Etape 2 : estimation de la pénombre (poids).
- Etape 3 : filtrage PCF avec une taille de kernel déterminée par l'étape 2.



$$w_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot w_{Light}}{d_{Blocker}}$$

PCSS – taille de la pénombre

- $d_{blocker}$: résultat de la recherche
- $d_{receiver}$: depth du pixel actuellement rendu
- w_{light} : dimensions de la source lumineuse

$$w_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot w_{Light}}{d_{Blocker}}$$

- Petite amélioration : on peut également prendre en compte la distance à la caméra

$$w_{penumbra} = \frac{(d_{receiver} - d_{blocker})}{d_{blocker}} * w_{light} * d_{observer}$$

- Où $d_{observer}$ = 1.0 / (distance^2 * bias)
  - Permet de rendre le kernel plus petit lorsque la distance à la caméra est large

Optimisation du PCSS

- La recherche des blockers est le principal bottleneck
- Pour une lumière directionnelle on peut utiliser un Hierarchical-Z (mipmap de depth buffer) pour accélérer la recherche (comme en ray marching) de $d_{blocker}$.

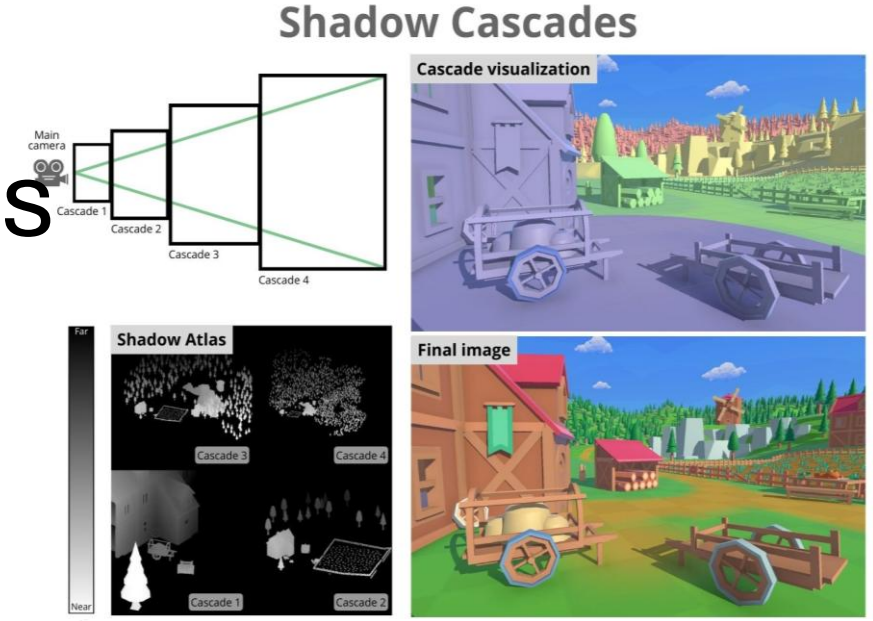
- Pour une lumière omnidirectionnelle on va en plus effectuer la recherche et le filtrage en screen-space

- Aller plus loin: prendre en compte des kernels anisotropiques
- Voir https://fr.slideshare.net/slideshow/04-shadows/48338416

Optimisation du PCSS

- La formulation "historique" utilise des nombres aléatoires tirés d'une distribution de Poisson.
  - Beaucoup d'échantillons nécessaires, filtre "fixe" (à recalculer si les paramètres changent)
- On connait des techniques type "blue noise" plus intéressantes : la distribution du disque de Vogel, et un générateur de nombre aléatoire comme "Interleaved Gradient Noise".
- La recherche de blockers peut également être améliorée par une meilleure génération d'échantillons plus proches des dimensions projetées de la source lumineuse.
- Combiné à un Hi-Z et quelques autres petites astuces on arrive à des performances très bonnes.
- https://wojtsterna.com/wp-content/uploads/2023/02/contact_hardening_soft_shadows.pdf

PSSM / Cascaded Shadow Maps



- Parallel-Split : L'idée consiste à splitter le domaine en plusieurs parties ou portions (parallèles) alignées sur le frustum de vue.

- Avantage:
  - possible d'avoir une résolution voire une fréquence de rafraichissement différente pour chaque split/cascade
  - Meilleure précision proche de la caméra

- Désavantage:
  - souvent plus coûteux du fait des multiples passes pour chaque cascade (sauf à réduire la fréquence)
  - Transitions et bordures entre les splits/cascade pas toujours précise/évidente

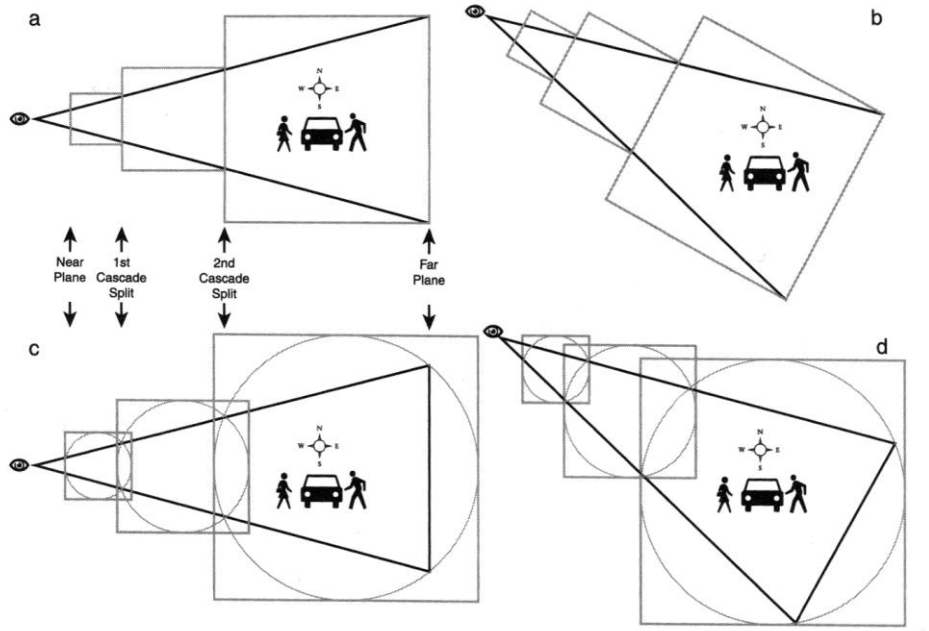
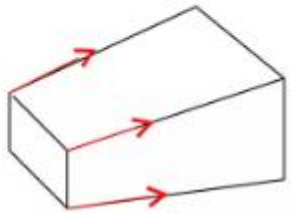
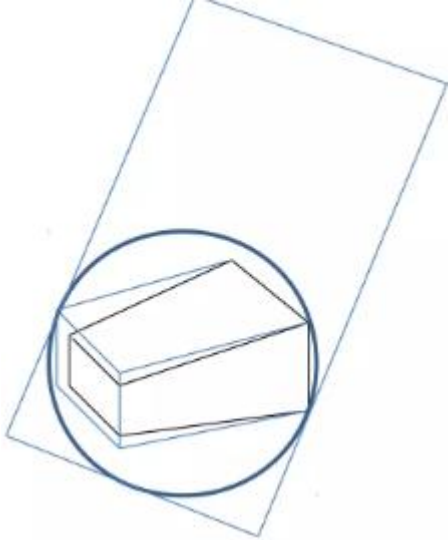


FIGURE 4.1.2 The view frustum in world space split into three cascade frustums and their corresponding shadow map coverage. We use a top view with the light direction pointing straight down the horizontal world plane.

```
// world space bound frustum directions
vec3 directions[4];
for(int i = 0; i < 4; i++)
      directions[i] = normalize(points[i + 4] - points[i]);
```



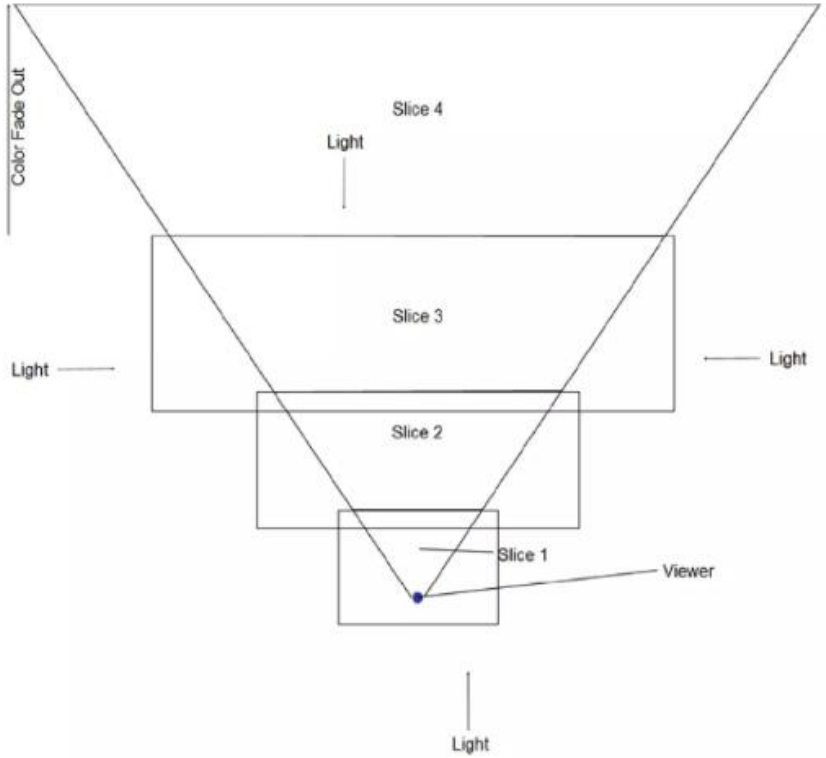


PSSM / CSM en pratique

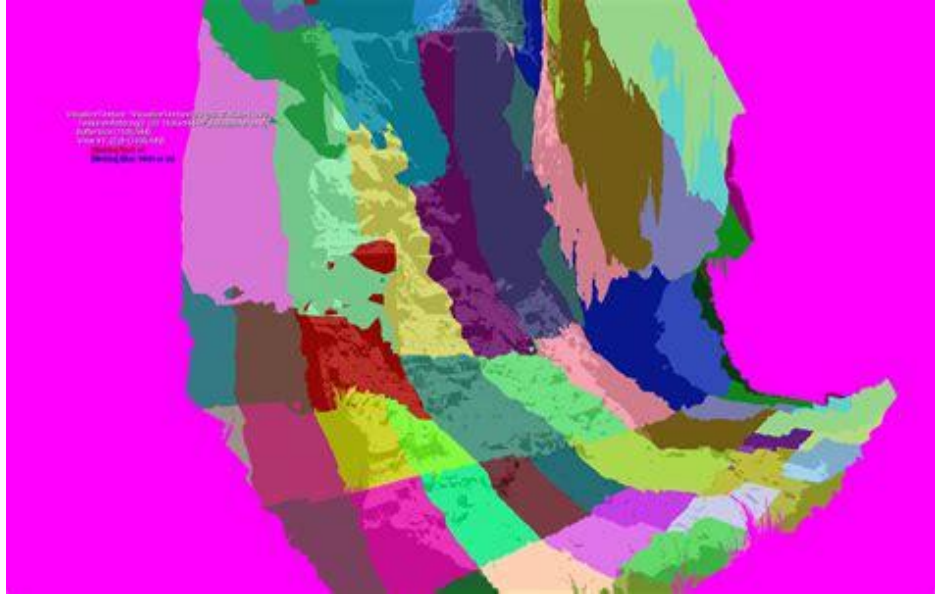
- Pour calculer un light volume le plus resserré possible:
  - On commence par calculer une AABB pour le view frustum qu'on inclut dans une sphère (calcul d'un radius).
  - On utilise ce radius pour déterminer les dimensions du volume de projection (cube orthographique).
  - Le near et le far dépendent de la distance max de visibilité de la light.

- Le plus compliqué reste le calcul des "splits" des light volumes en plusieurs régions.
- Plus de détail ici, et dans les articles de GPU Gems 2 et Shader X 5 et 6

https://fr.slideshare.net/slideshow/04-shadows/48338416



Virtual Shadow Maps (VSM)

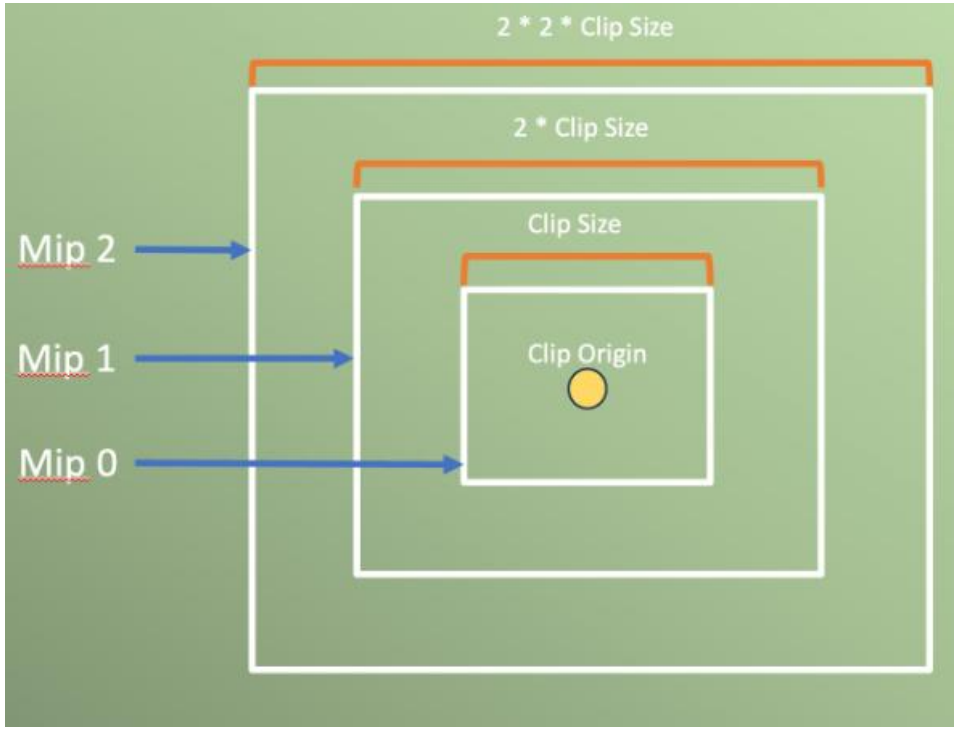


- *A ne pas confondre avec Variance Shadow Map.*
- Technique actuellement utilisée par UE4/UE5.
- Mélange 2 concepts connus : Virtual Textures et Clip Maps.
- Virtual Textures : on scinde les textures en pleins de petits morceaux qu'on stock dans des atlas (parfois avec du streaming).
- Idée déjà exploitée pour les Shadow Maps :

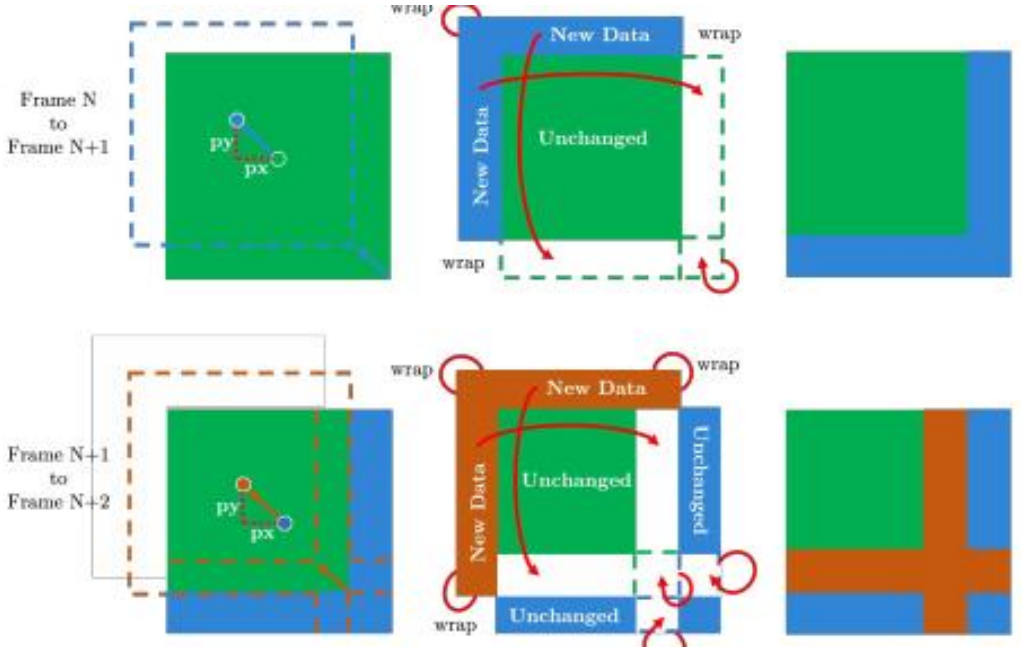
https://www.cg.tuwien.ac.at/research/publications/2007/GIEGL-2007-FVS/GIEGL-2007-FVS-Preprint.pdf

Clip Maps

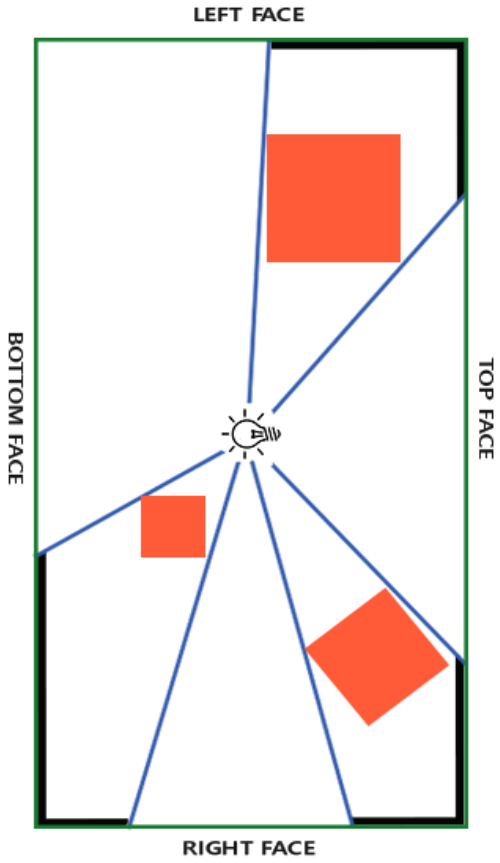
- Plusieurs anneaux (ring) concentriques en cascade.
  - Historiquement les Clip Maps peuvent être vues comme une extension des Mip Maps …
    - En théorie toutes les textures ont les mêmes dimensions.
    - On représente en fait la même surface mais à une distance (LoD) différente.



- Dans le cas VSM, il s'agit bien de textures de tailles différentes, qui représentent des anneaux couvrants une distance plus large.
- Fonctionnement toroïdal : lorsque la caméra (clip origin) se déplace, on met à jour seulement une portion de la clip map, en L ici, de manière circulaire (wrapping).
  - Un LoD de clip map contient donc des portions récentes et d'autres plus anciennes mais virtuellement proches.



Cas des lumières omnidirectionnelles

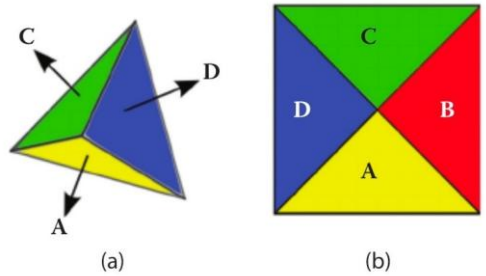


- On utilise généralement des cubemaps pour stocker les ombres. https://learnopengl.com/Advanced-Lighting/Shadows/Point-Shadows

- Une optimisation: checker si le frustum d'une face de cubemap est visible depuis le view frustum (caméra).
  - Le frustum des faces est créé par extrusion des bounds des casters.
  - Calcul d'intersection frustum/frustum, on skip les faces inutiles.

- Dans le cas des cubemaps locales, on peut utilisé la technique de Parallax Cubemap pour simuler des soft shadows statiques: https://community.arm.com/arm-community-blogs/b/mobile-graphics-and-gaming-blog/posts/dynamic-soft-shadows-based-on-local-cubemap

- On peut aussi utiliser d'autres formes (octaèdre, tetraèdre...).



Ambient shadowing

- Une majeure partie des ombres provient de l'ambiance.
- Donc en relation avec l'illumination globale et indirecte.

- Ambient Occlusion: approximations screen-space (SSAO, SSDO)

- Extension des CHS (Contact Harderning) : on utilise de vrais rayons dans le repère de la shadow map (ray marching).
  - Ce ray marching peut se faire seulement en screen-space car on ne s'intéresse qu'aux pixels illuminés à l'écran.

- Voir aussi la méthode utilisée dans Days Gone et d'autres titres ici:

https://www.bendstudio.com/blog/inside-bend-screen-space-shadows

Aller plus loin

- Deep Shadow Maps (encore trop lourd pour du temps réel)

https://graphics.stanford.edu/papers/deepshadows/deepshad.pdf

- Shadow Map dédiées aux personnages (cinématiques) et cheveux

https://www.cemyuksel.com/research/deepopacity/

https://www.cse.chalmers.se/~uffe/hair2009.pdf

https://forums.ea.com/discussions/dragon-age-the-veilguard-discussion-en/innovating-strand-by-strand-for-lifelike-hair-in-dragon-age-the-veilguard/5016649

- Irregular Z-Buffer et Frustum Traced Shadows

https://research.nvidia.com/sites/default/files/pubs/2015-02_Frustum-Traced-Raster-Shadows/ftizb_cameraReady.pdf

- (Hybrid) Frustum-Traced Shadows

https://www.nvidia.com/en-us/geforce/news/nvidia-hfts-hybrid-frustum-traced-shadows/