# IFT3355: Infographie
# **Sujet 5**: `shading` 3
# (illumination *locale* 3)

Derek Nowrouzezahrai
Département d'informatique et de recherche opérationelle
Université de Montréal

# *Shadings*

## (à ne pas confondre avec *Shading*)

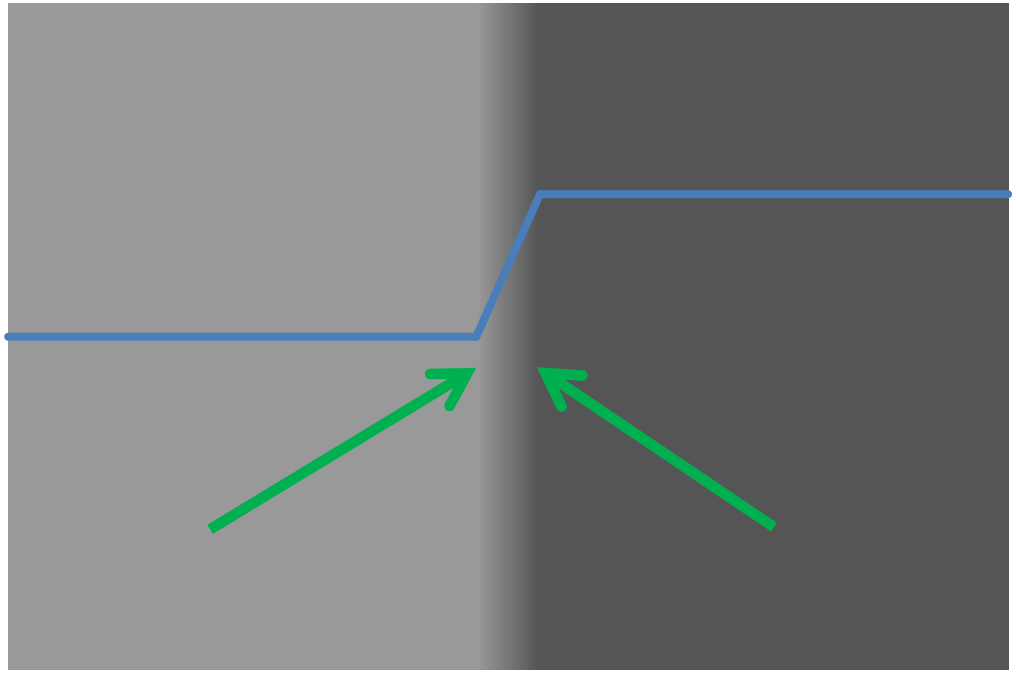
- On peut calculer l'illumination en chaque point d'une scène, mais il est possible d'en réduire le coût en utilisant des approximations pour l'illumination sur un polygone
- Plusieurs techniques:
  - *Flat shading*
  - *Gouraud shading*
  - *Phong shading*
    - (à ne pas confondre avec le modèle d'illumination *Phong*)

# Flat Shading

- calcule l'illumination pour un point du polygone (centre ou un sommet)
- suppose que cette illumination est la même sur tout le polygone
- couleur uniforme et *Mach bands*
- valide si
  - $\left(\vec{N} \cdot \vec{L}\right)$ est constant : lumière directionnelle
  - $\left(\vec{N} \cdot \vec{E}\right)$ est constant : projection parallèle

# Mach bands

- Une discontinuité de valeur ou de pente d'intensité est exagérée perceptuellement par l'oeil dû à un facteur d'inhibition de récepteurs adjacents dans l'oeil



# Calcul de normales des polygones

- Calcul analytique si on connaît la surface que le polygone approxime (ex: sphère)
- Moyenne des normales des polygones dont le sommet fait partie (ou moyenne pondérée par l'angle formé par ce polygone au sommet)

$$\vec{N} = \frac{\sum \vec{N}_i}{\left\| \sum \vec{N}_i \right\|}$$

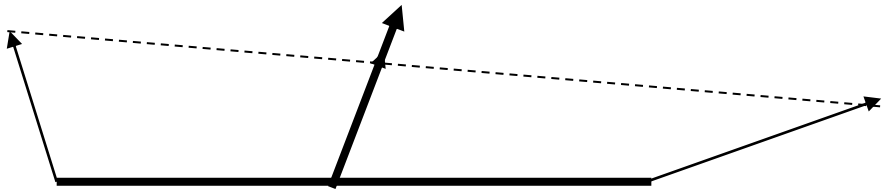
- Si une arête existe vraiment, un sommet a *i* normales dont une seule est choisie dépendant du polygone à rendre

# Shading de Gouraud

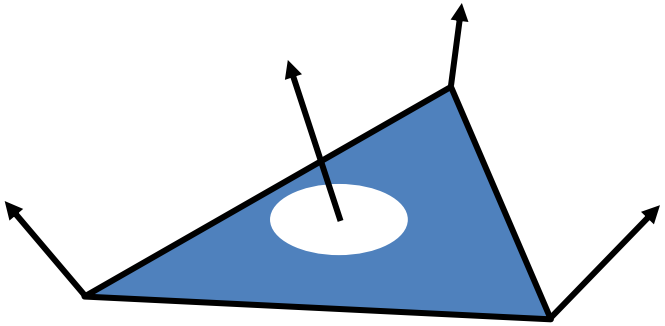
- calcule l'illumination à chaque sommet du polygone (normale + illumination → couleur)
- interpolation bilinéaire des couleurs tout comme on le faisait pour la profondeur
- rapide mais peut rater des *highlights,* il faut alors un maillage plus fin des polygones
- élimine les discontinuités d'intensité mais pas celles de pente d'intensité, donc les *Mach bands* sont réduites mais elles sont toujours présentes

# *Shading* de Phong

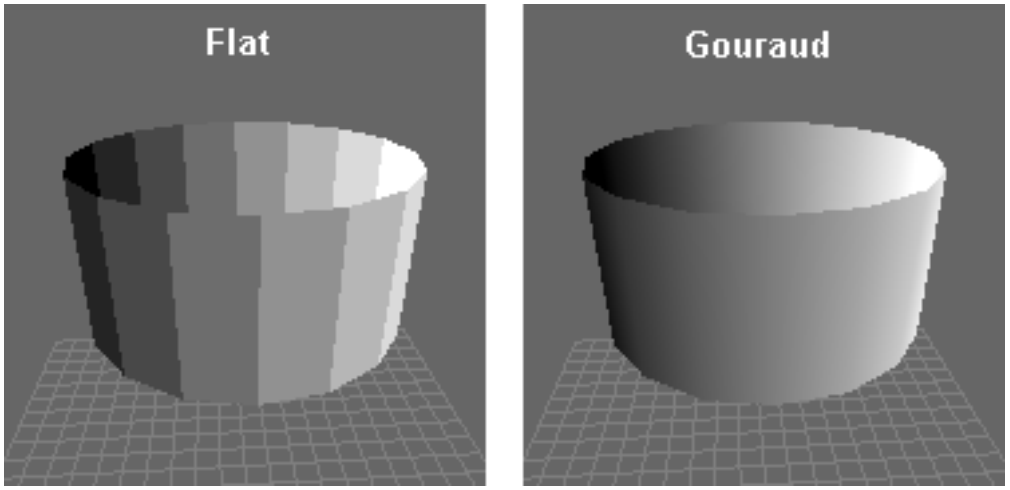
- interpolation bilinéaire des normales pour tout point dans le polygone
- calcule l'illumination pour chaque normale interpolée

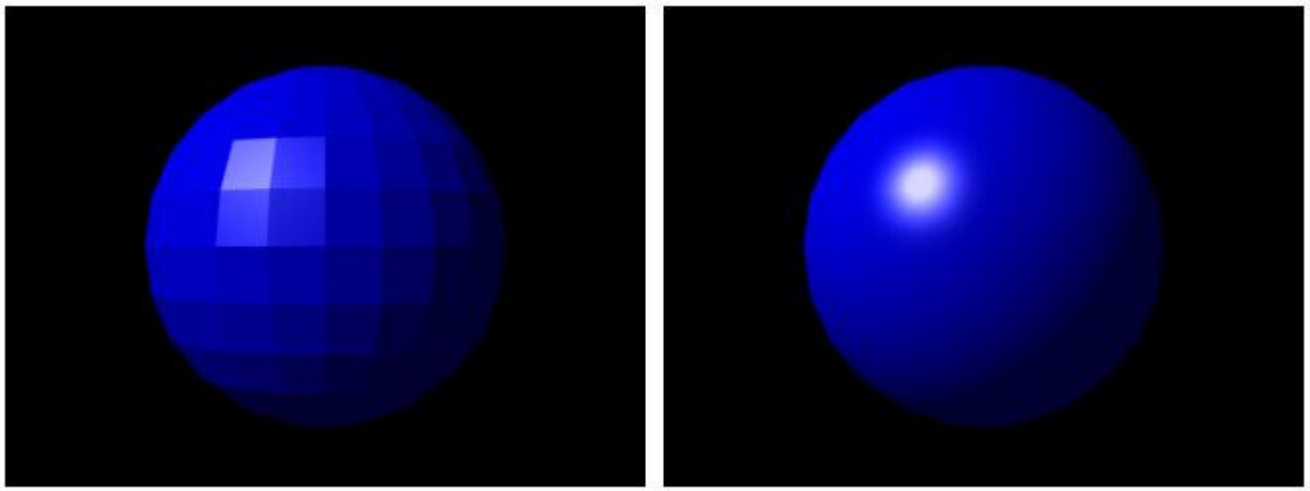


- Plus coûteux à calculer mais approxime beaucoup mieux les *highlights*



# Flat vs. Gouraud vs. Phong



FLAT SHADING PHONG SHADING

# Survol: les ombres (ponctuelles)… encore

- Rappelons qu'on a déjà étudié comment effectuer le calcul des ombres causés par des lumières ponctuelle/directionnelles dans un système de **lancer de rayons**
- Avant étudier des effets plus compliqués d'illumination globale (plus ou moins dans un contexte de lancer de rayons), nous revisiterons le calcul des ombres ponctuelle (ombre dur) dans un système de **rastérisation**

- Nous étudierons deux techniques:
  - Un qui utilise un tampon de profondeur appelé un *shadow map*
  - Et un qui génère de la géométrie supplémentaires qui représentes des volume d'ombre – des *shadow volumes*

# Tampon de profondeur d'ombre - *Shadow map*

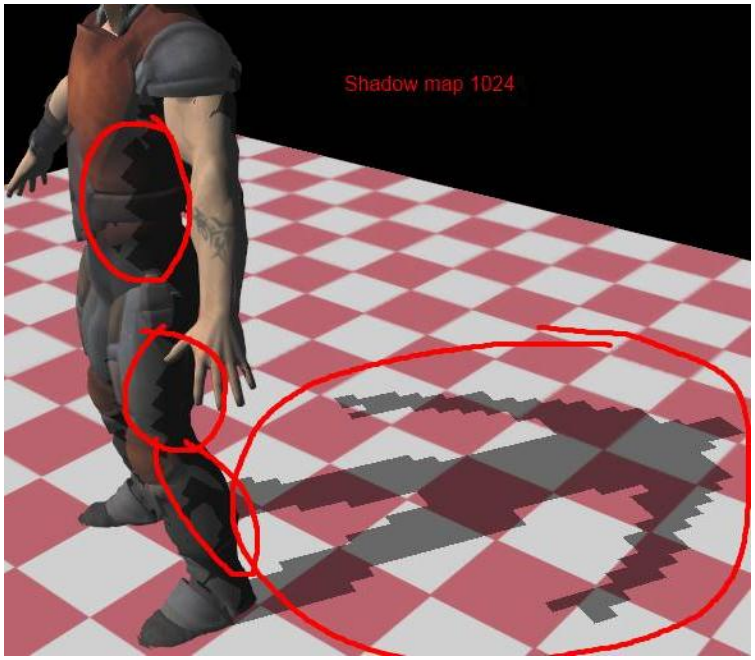
- Construit un tampon de profondeur à partir de la lumière *L* (aucun tampon image nécessaire)
- Pour déterminer si un point *P* est dans l'ombre

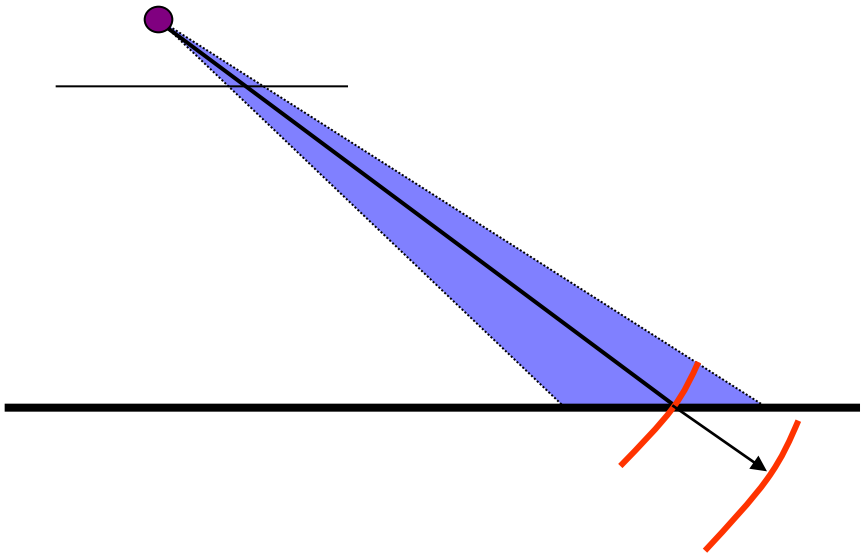
```
bool dans_ombre = true;
for( chaque pixel P sur l'écran )
  point P_l = mul(L_projection, P.position)
    if( distance( L – P ) < P_l.z )
      dans_ombre = false;
```

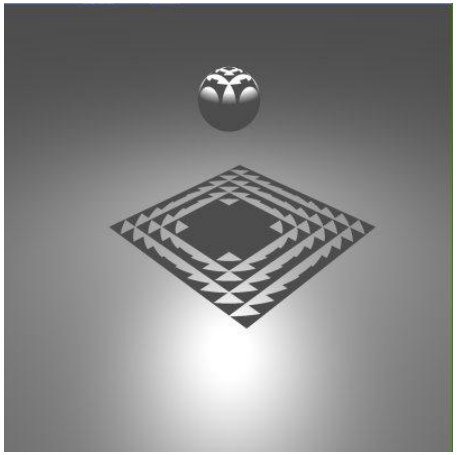
+ facile à implanter et disponible en hardware
- plusieurs *shadow maps* pour omni-directionnel
- résolution vs. mémoire
- aliassage

# Shadow map - artéfacts

- effet d'escaliers aux silhouettes des ombres
- *acné de surface* (ajoute epsilon à la distance du point)

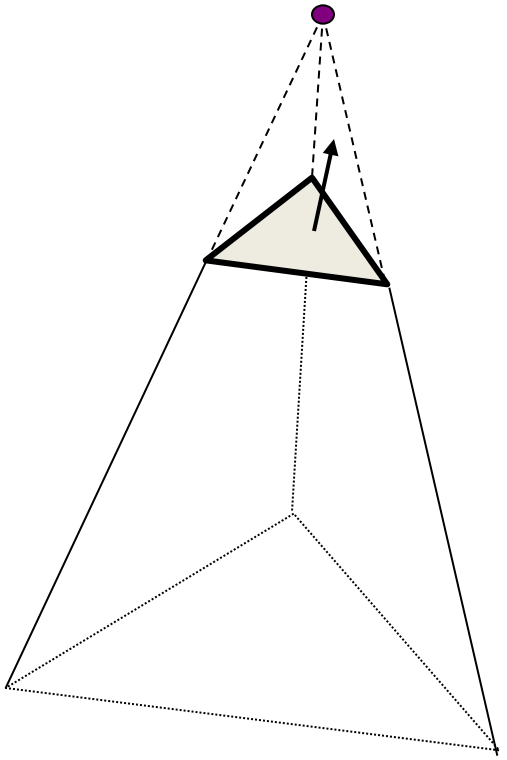


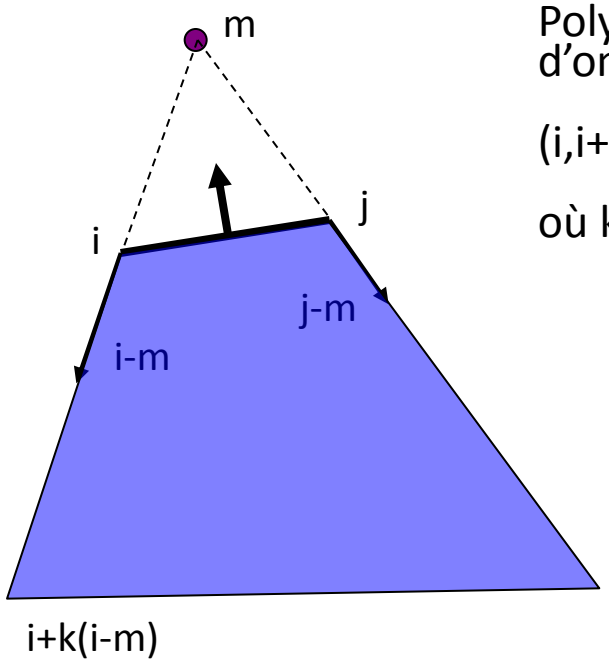





# Volumes d'ombre (de Crow)

- Pour chaque arête d'un polygone faisant face à la lumière, on construit un quadrilatère reposant sur le plan arête-lumière.






Polygone (quadrilatère) d'ombre:

(i,i+k(i-m),j+k(j-m),j)

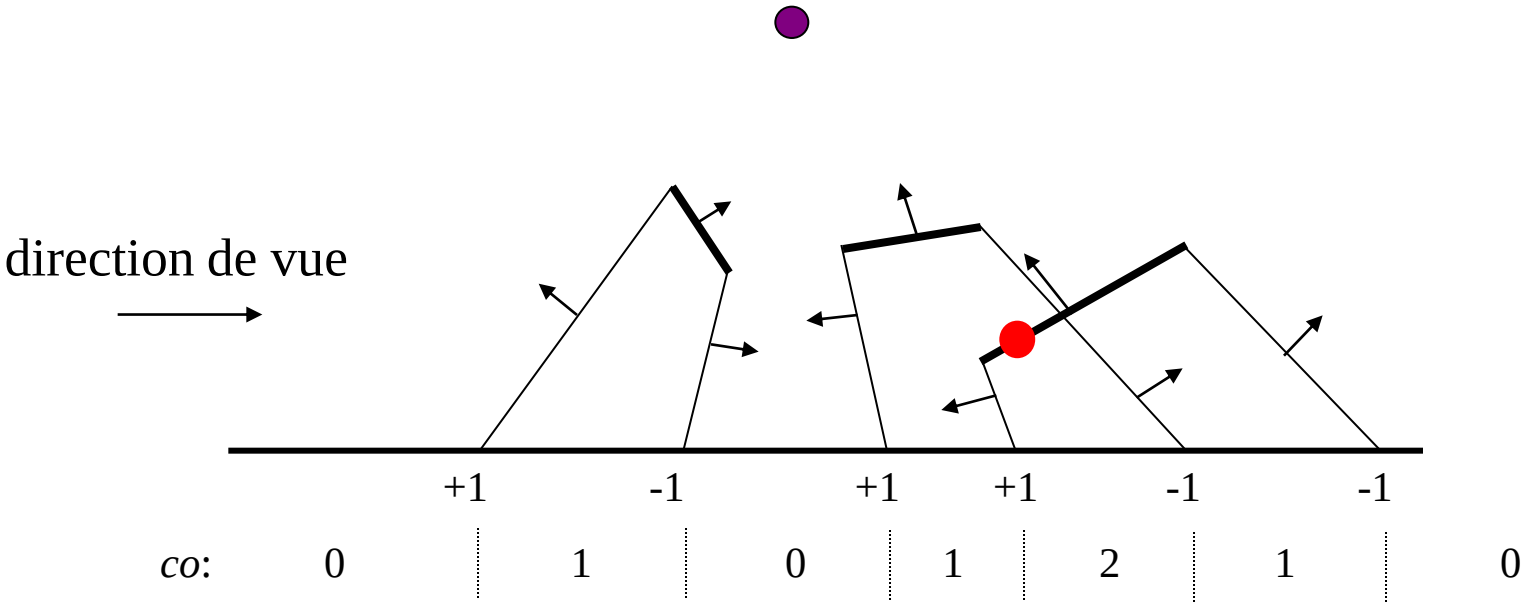
où k est très grand

# Volumes d'ombre

- Un point sera dans l'ombre s'il est à l'intérieur d'un polyèdre d'ombre défini par chaque polygone, sa projection de la lumière (*cap*) et ses polygones d'ombre
- On fait d'abord un rendu pour remplir le tampon de profondeur (*depth buffer*), puis un rendu des polygones d'ombre
- Lors de ce rendu, on conserve un compteur d'ombre *co*

```
if (N · E < 0)     // polygone d'ombre fait face à l'oeil
    co = co + 1    // front-facing
else
    co = co - 1    // back-facing
```

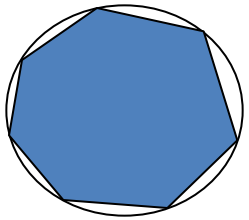
# Volumes d'ombre



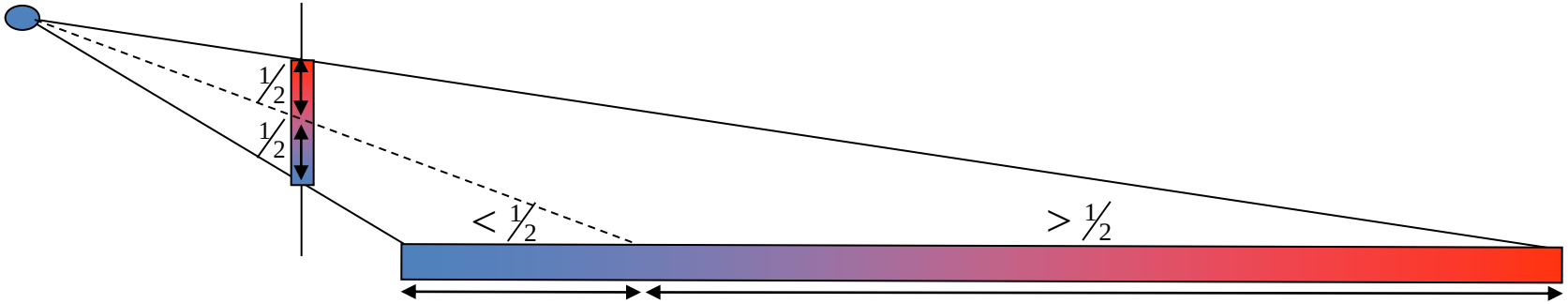

+ espace objet et omni-directionel
- seulement pour des polygones
- limites (min-max) sur les valeurs du compteur (*stencil planes*)
- initialisation de *co* pour le point de vue avec problèmes si les *front ou back clipping* intersectent un polygone d'ombre

# Problèmes reliés à l'interpolation du *shading*

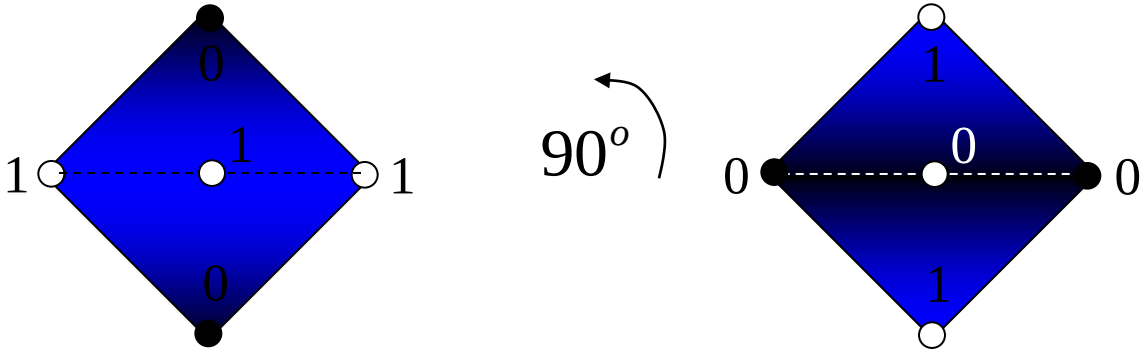
- Silhouette polygonale



- Distorsion due à la projection en perspective

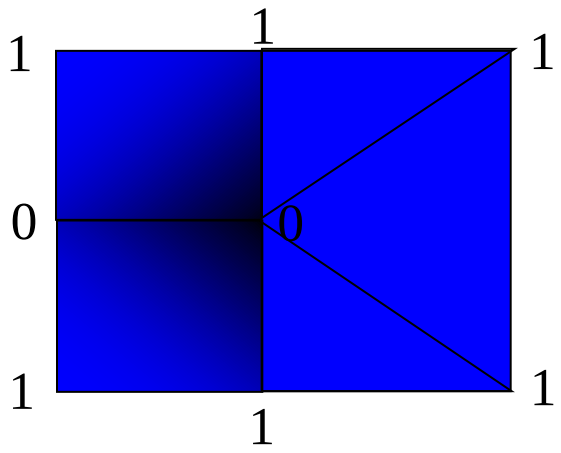



- Dépendance d'orientation



# Problèmes reliés à l'interpolation du *shading*

- Sommet en T




- Normales comme une mauvaise représentation