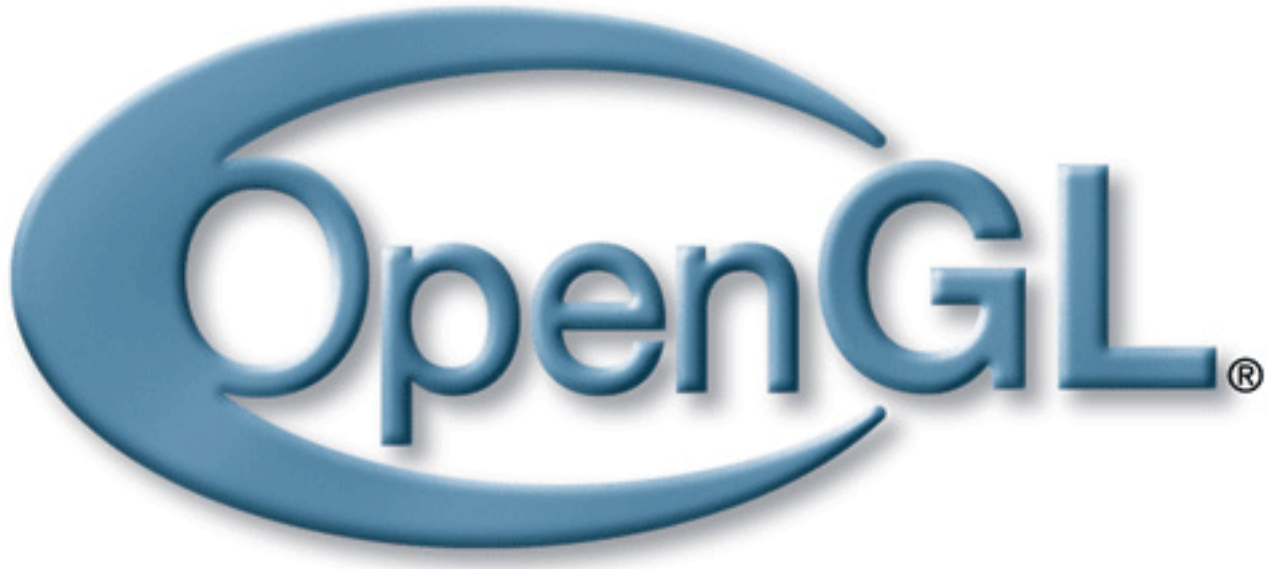
OpenGL - Teil 3

- Schatten mit OpenGL -



Bernhard Bittorf und Andy Reimann

OpenGL - Warum Schatten?

- bilden Realität besser ab
- Tiefeneinschätzung

OpenGL - eine reiner Rasterisierer

- rendert nur Dreiecke
- keine Build In Funktionalität für Schatten
- genug Funktionalität, um Schatten zu implementieren
- verschiedene Buffer (Tiefenbuffer, Stencilbuffer, Accumulationbuffer) zum Speichern von Informationen *pro Pixel*

Schatten Techniken im Überblick

Plane Projected Shadows

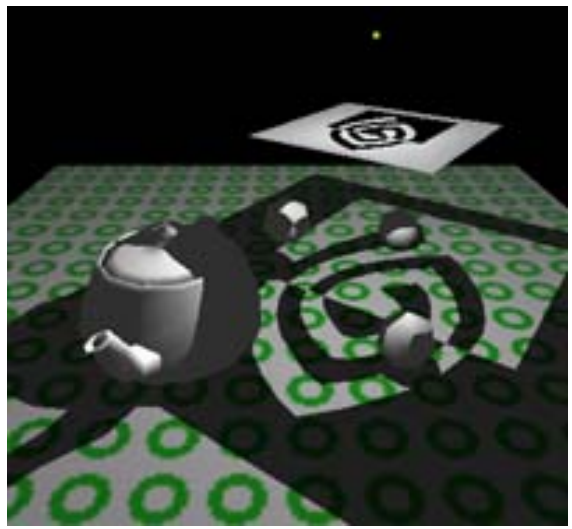
- schell
- detailiert
- keine Selbstschattierung
- keine Objektschattierung

Schatten Techniken im Überblick

Plane Projected Shadows

- schell
- detailiert
- keine Selbstschattierung
- keine Objektschattierung

Projected Shadows



- □schell
- Detail hängt von Textur ab
- keine Selbstschattierung
- Objektschattierung

OpenGL - Die Techniken im Überblick

Shadow Mapping

- sehr schnell
- Detail hängt von Textur ab
- Selbstschattierung
- Objektschattierung
- Soft-Shadows möglich

OpenGL - Die Techniken im Überblick

Shadow Mapping

- sehr schnell
- Detail hängt von Textur ab
- Selbstschattierung
- Objektschattierung
- Soft-Shadows möglich

Shadow Volumes

- langsam (hängt von der Geometrie-Komplexität ab)
- hohes Detail (Pixelgenau)
- Selbstschattierung
- Objektschattierung
- keine Soft-Shadows möglich (Ansatz: mehrere Lichtquellen beieinander = Arealight, sehr langsam)

OpenGL - Die Techniken im Überblick

Shadow Mapping

- sehr schnell
- Detail hängt von Textur ab
- Selbstschattierung
- Objektschattierung
- Soft-Shadows möglich

Nächste Aufgabe

Shadow Volumes

- langsam (hängt von der Geometrie-Komplexität ab)
- hohes Detail (Pixelgenau)
- Selbstschattierung
- Objektschattierung
- keine Soft-Shadows möglich (Ansatz: mehrere Lichtquellen beieinander = Arealight, sehr langsam)

OpenGL - Shadow Mapping

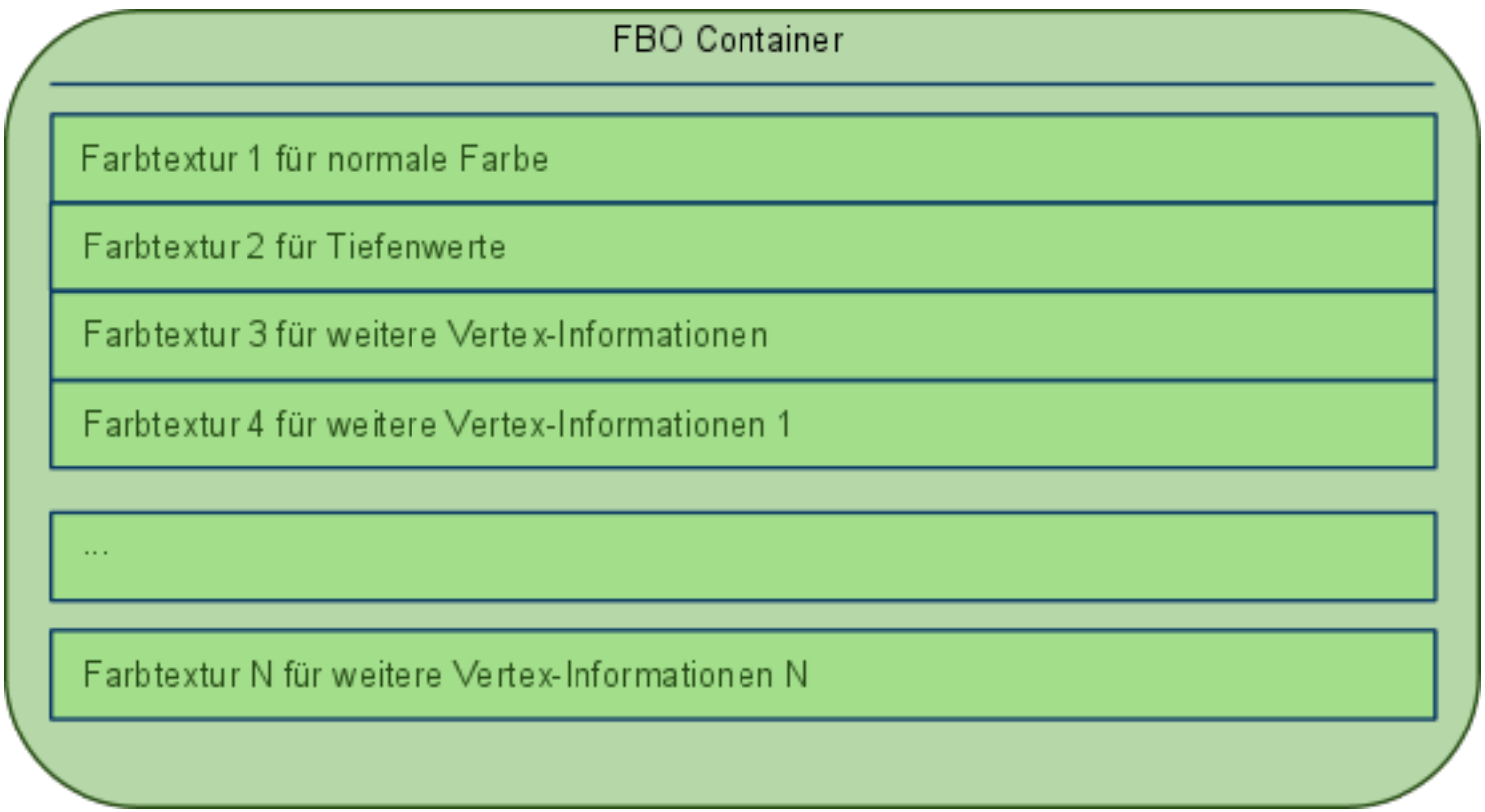
- Textur basiert (RTT - Render to Texture)
- 3 Render Durchläufe:
  - Szene mit ambienter Beleuchtung rendern
  - Szene aus Sicht der Lichtquelle in eine Tiefentextur rendern
  - Szene beleuchtet mit Tiefentest in OpenGL rendern

OpenGL - Render-to-Texture

- FrameBufferObject (FBO)
  - effizient in eine Textur rendern
  - relativ einfach
  - Container für Tiefenbuffer/Farbbuffer/...
  - kann als *Rendertarget* verwendet werden
  - *Offscreen Rendering*

OpenGL - Render-to-Texture

- FrameBufferObject (FBO)
  - effizient in eine oder mehrere Textur rendern
  - relativ einfach
  - Container für Tiefenbuffer/Farbbuffer/...
  - Kann als *Rendertarget* verwendet werden
  - *Offscreen Rendering*



OpenGL - Render-to-Texture

1. Ein neues FBO erzeugen

```
unsigned width = 1024;
unsigned height = 1024;
// ein neues FBO erstellen
GLuint fbo;
glGenFramebuffersEXT(1, &fbo);
// einen Tiefenbuffer hinzufügen
GLuint depthbuffer;
glGenRenderbuffersEXT(1, &depthbuffer);
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depthbuffer);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT, width, height);
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_RENDERBUFFER_EXT, depthbuffer);
```

OpenGL - Render-to-Texture

1. Ein neues FBO erzeugen

```
unsigned width = 1024;
unsigned height = 1024;
// ein neues FBO erstellen
GLuint fbo;
glGenFramebuffersEXT(1, &fbo);
// einen Tiefenbuffer hinzufügen
GLuint depthbuffer;
glGenRenderbuffersEXT(1, &depthbuffer);
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depthbuffer);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT, width, height);
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_RENDERBUFFER_EXT, depthbuffer);
```

Nicht unbedingt benötigt

OpenGL - Render-to-Texture

1. Ein neues FBO erzeugen

```
unsigned width = 1024;
unsigned height = 1024;
// ein neues FBO erstellen
GLuint fbo;
glGenFramebuffersEXT(1, &fbo);
// einen Tiefenbuffer hinzufügen
GLuint depthbuffer;
glGenRenderbuffersEXT(1, &depthbuffer);
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depthbuffer);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT, width, height);
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_RENDERBUFFER_EXT, depthbuffer);
```

2. Eine Textur hinzufügen

```
// eine neue Textur erzeugen
GLuint img;
glGenTextures(1, &img);
glBindTexture(GL_TEXTURE_2D, img);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8,  width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, NULL);
// die textur an das FBO hängen
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, img, 0);
```

Kann eine Farb- oder Tiefentextur sein

OpenGL - Render-to-Texture

1. Ein neues FBO erzeugen

```
unsigned width = 1024;
unsigned height = 1024;
// ein neues FBO erstellen
GLuint fbo;
glGenFramebuffersEXT(1, &fbo);
// einen Tiefenbuffer hinzufügen
GLuint depthbuffer;
glGenRenderbuffersEXT(1, &depthbuffer);
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depthbuffer);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT, width, height);
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_RENDERBUFFER_EXT, depthbuffer);
```

2. Eine Textur hinzufügen

```
// eine neue Textur erzeugen
GLuint img;
glGenTextures(1, &img);
glBindTexture(GL_TEXTURE_2D, img);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8,  width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, NULL);
// die textur an das FBO hängen
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, img, 0);
```

3. Das FBO auf Fehler überprüfen

```
// WICHTIG: auf fehler überprüfen
GLenum status = glCheckFramebufferStatusEXT(GL_FRAMEBUFFER_EXT);
```

OpenGL - Render-to-Texture

1. Ein neues FBO erzeugen

```
unsigned width = 1024;
unsigned height = 1024;
// ein neues FBO erstellen
GLuint fbo;
glGenFramebuffersEXT(1, &fbo);
// einen Tiefenbuffer hinzufügen
GLuint depthbuffer;
glGenRenderbuffersEXT(1, &depthbuffer);
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depthbuffer);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT, width, height);
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_RENDERBUFFER_EXT, depthbuffer);
```

2. Eine Textur hinzufügen

```
// eine neue Textur erzeugen
GLuint img;
glGenTextures(1, &img);
glBindTexture(GL_TEXTURE_2D, img);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8,  width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, NULL);
// die textur an das FBO hängen
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, img, 0);
```

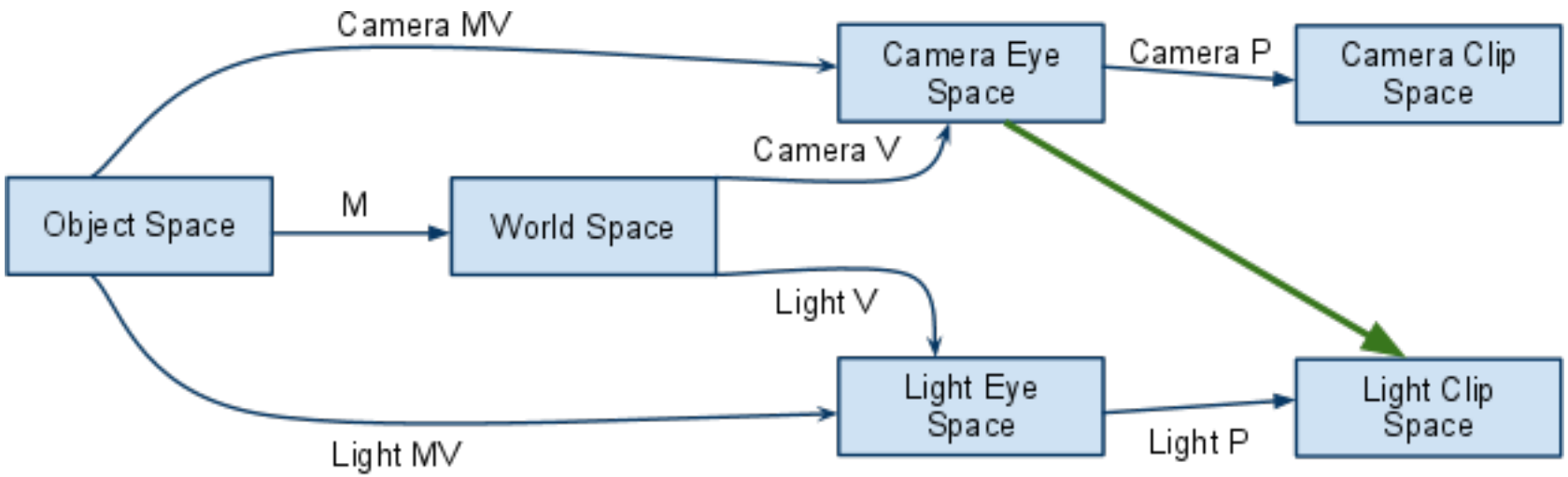
3. Das FBO auf Fehler überprüfen

```
// WICHTIG: auf fehler überprüfen
GLenum status = glCheckFramebufferStatusEXT(GL_FRAMEBUFFER_EXT);
```

korrekte Fehlerabfrage: http://wiki.delphigl.com/index.php/Tutorial_Framebufferobject

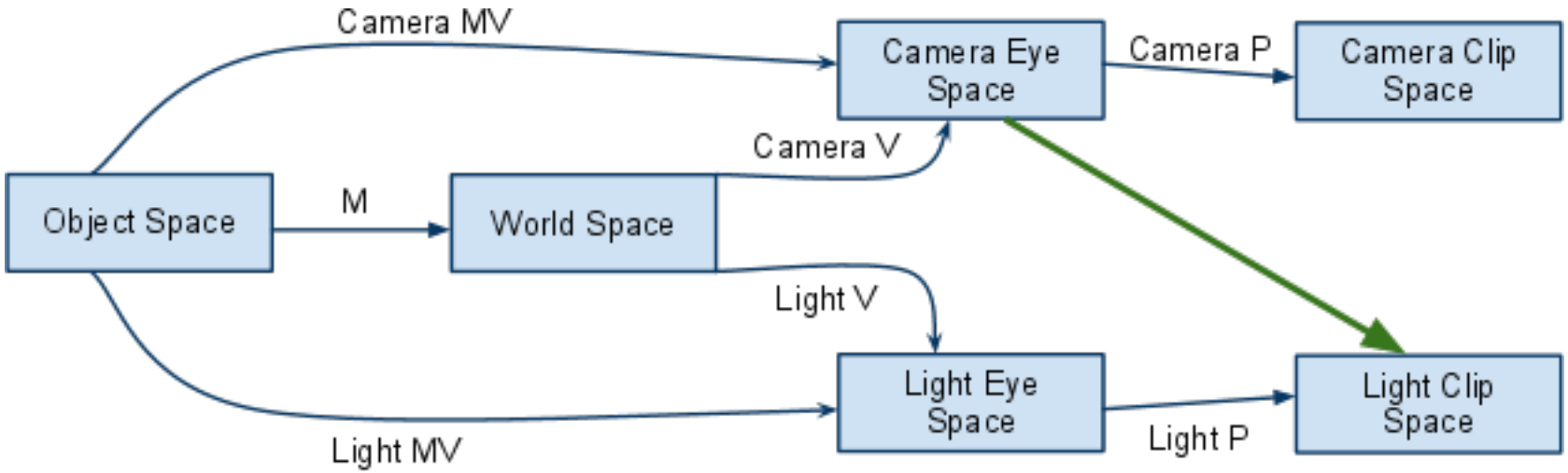
OpenGL - zurück zu Shadow Mapping

- theoretisch nur ein Tiefentest im *Light Clip Space*
  - *Was sieht das Licht an diesem Pixel für eine Tiefe?*
- *Camera Space* in *Light Clip Space* für Tiefentest



M = Modelmatrix, V = Viewmatrix, MV = Modelviewmatrix, P = Projectionmatrix

OpenGL - zurück zu Shadow Mapping



M = Modelmatrix, V = Viewmatrix, MV = Modelviewmatrix, P = Projectionmatrix

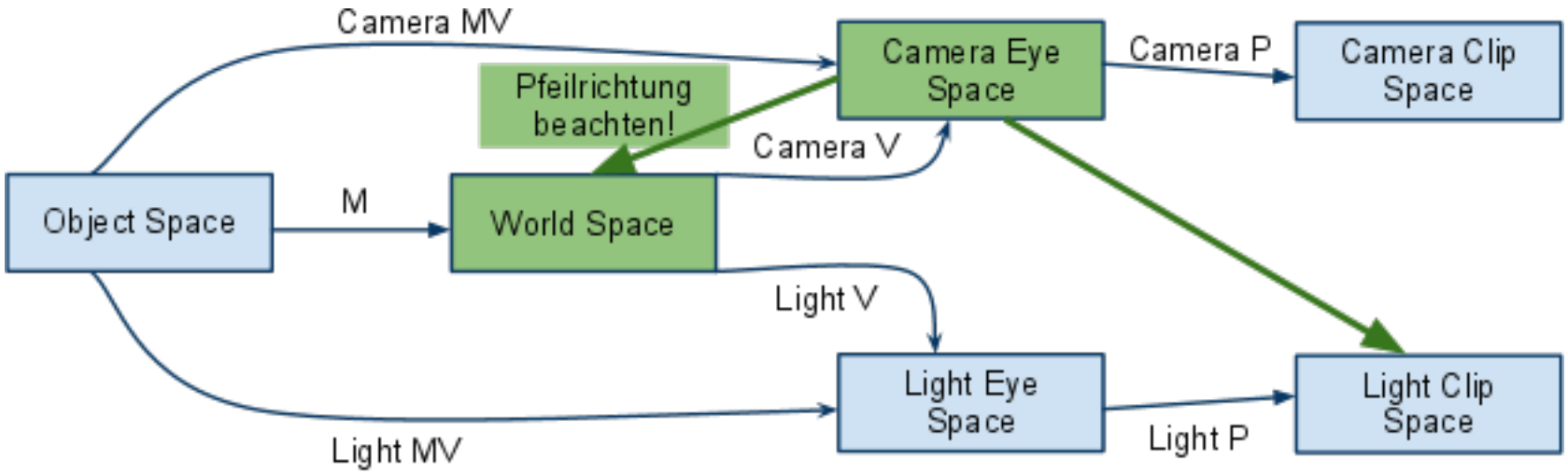
Hinweis: Matrizenmultiplikation von rechts nach links lesen

$$[TexMatrix] = [Bias] \times [LightP] \times [LightV] \times [CameraV]^{-1}$$

$$[Bias] = \begin{pmatrix} 0.5 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

- Skalierung & Translation von [-1,1] nach [0,1] (Texturkoordinaten)

OpenGL - zurück zu Shadow Mapping



M = Modelmatrix, V = Viewmatrix, MV = Modelviewmatrix, P = Projectionmatrix

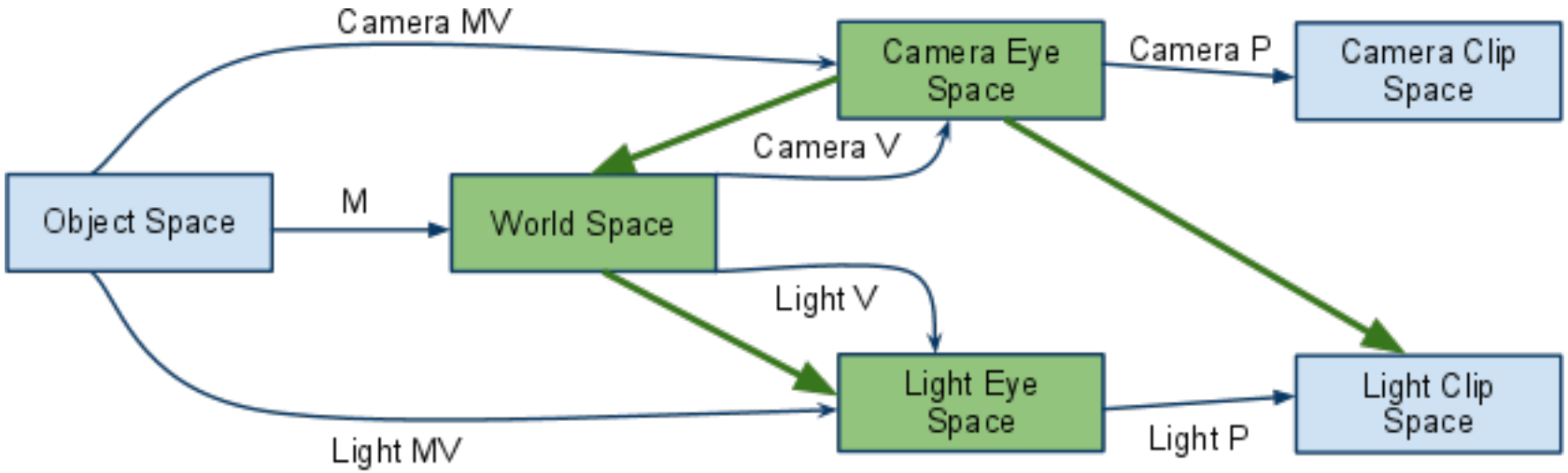
Hinweis: Matrizenmultiplikation von rechts nach links lesen

$$[TexMatrix] = [Bias] \times [LightP] \times [LightV] \times [CameraV]^{-1}$$

$$[Bias] = \begin{pmatrix} 0.5 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

- Skalierung & Translation von [-1,1] nach [0,1] (Texturkoordinaten)

OpenGL - zurück zu Shadow Mapping



M = Modelmatrix, V = Viewmatrix, MV = Modelviewmatrix, P = Projectionmatrix

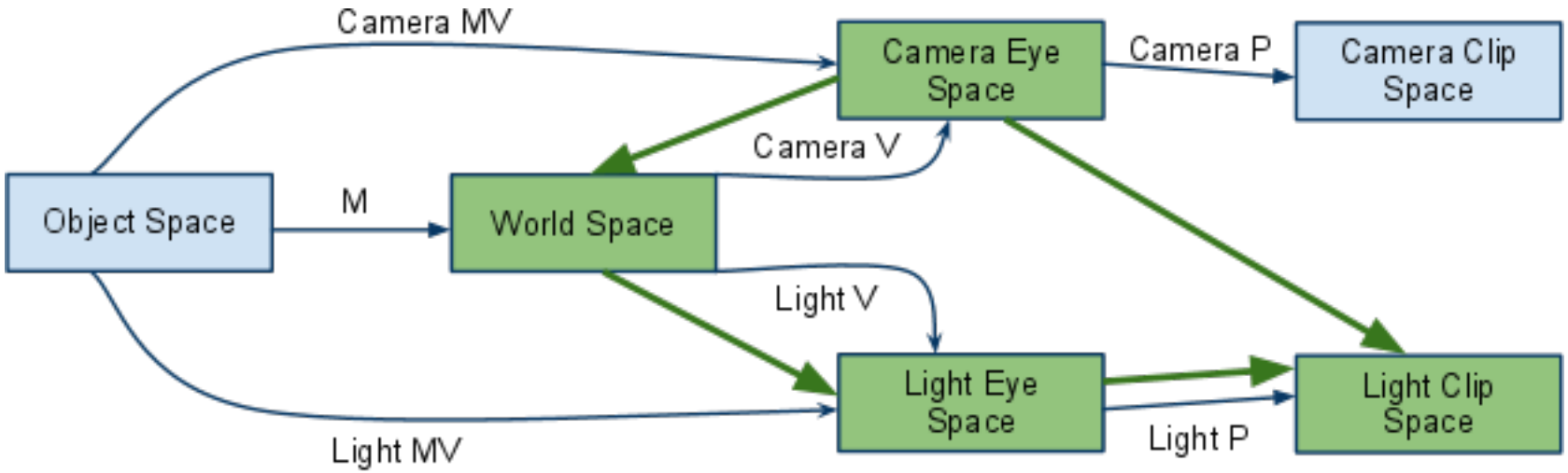
Hinweis: Matrizenmultiplikation von rechts nach links lesen

$$[TexMatrix] = [Bias] \times [LightP] \times [LightV] \times [CameraV]^{-1}$$

$$[Bias] = \begin{pmatrix} 0.5 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

- Skalierung & Translation von [-1,1] nach [0,1] (Texturkoordinaten)

OpenGL - zurück zu Shadow Mapping



M = Modelmatrix, V = Viewmatrix, MV = Modelviewmatrix, P = Projectionmatrix

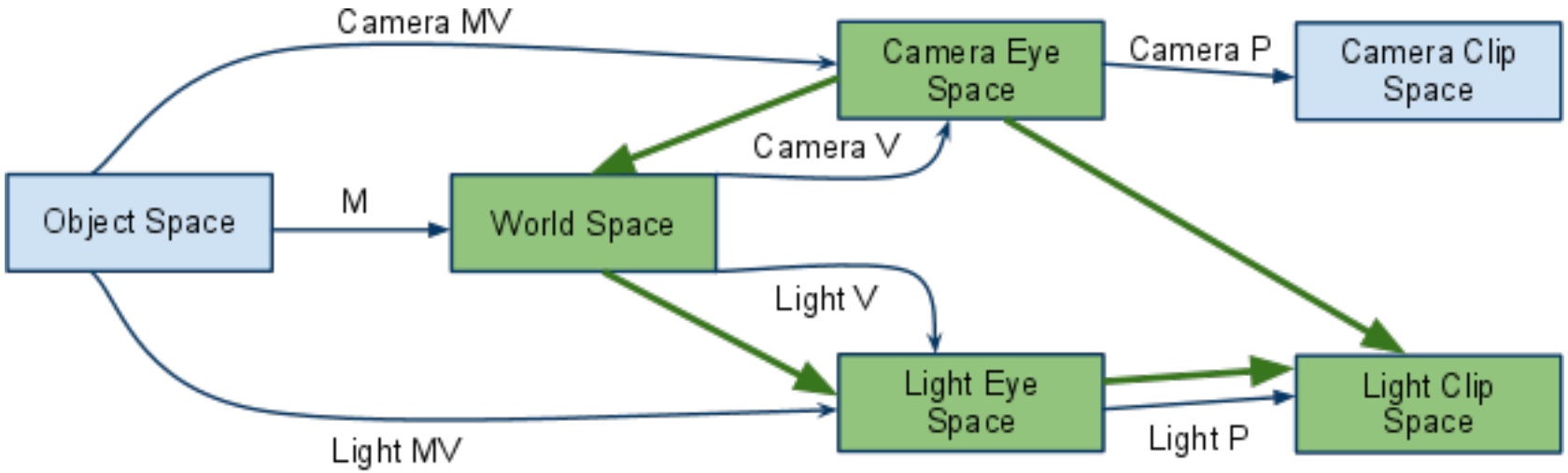
Hinweis: Matrizenmultiplikation von rechts nach links lesen

$$[TexMatrix] = [Bias] \times [LightP] \times [LightV] \times [CameraV]^{-1}$$

$$[Bias] = \begin{pmatrix} 0.5 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

- Skalierung & Translation von [-1,1] nach [0,1] (Texturkoordinaten)

OpenGL - zurück zu Shadow Mapping



M = Modelmatrix, V = Viewmatrix, MV = Modelviewmatrix, P = Projectionmatrix

Hinweis: Matrizenmultiplikation von rechts nach links lesen

$$[TexMatrix] = [Bias] \times [LightP] \times [LightV] \times [CameraV]^{-1}$$

CameraV muss weggelassen werden, wenn man Textur-Koordinaten-Generierung (nächste Folie) verwendet. Nur wenn man im GL_TEXTURE Matrix Mode mit glLoadMatrix die TexMatrix verwenden will ist dies notwending, da hier die Kamera nicht automatisch mit eingerechnet wird!

OpenGL - zurück zu Shadow Mapping

- Matrix auf Texturkoordinaten anwenden
  - Texturkoordinatengenerierungsfunktion von OpenGL
  - Referenz von Depth Map zu Weltkoordinaten hergestellt

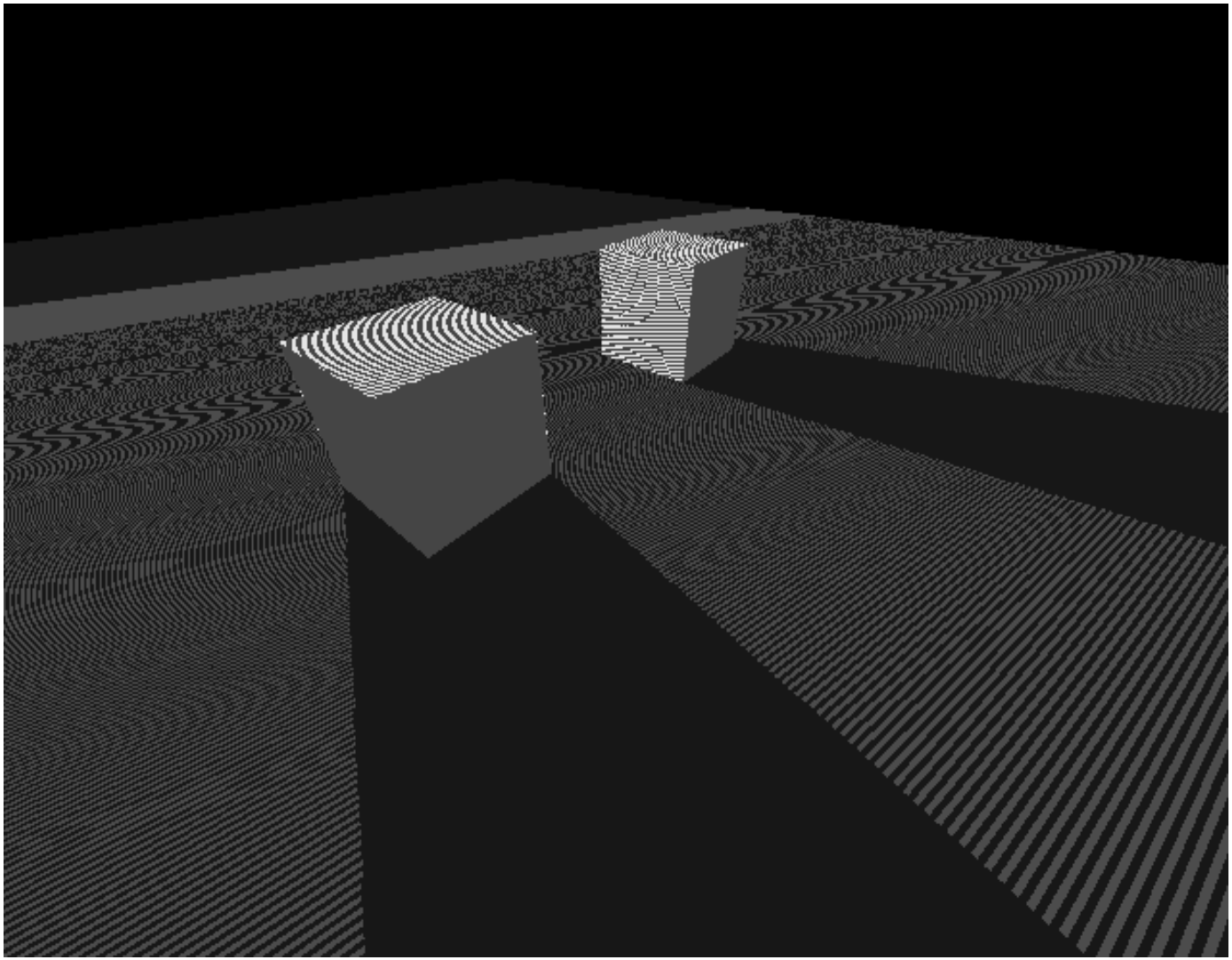
```
Mat4 texMat; // do some calculation here
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGenfv(GL_S, GL_EYE_PLANE, texMat.getRow(0));
glEnable(GL_TEXTURE_GEN_S);

glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGenfv(GL_T, GL_EYE_PLANE, texMat.getRow(1));
glEnable(GL_TEXTURE_GEN_T);

glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGenfv(GL_R, GL_EYE_PLANE, texMat.getRow(2));
glEnable(GL_TEXTURE_GEN_R);

glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGenfv(GL_Q, GL_EYE_PLANE, texMat.getRow(3));
glEnable(GL_TEXTURE_GEN_Q);
```

OpenGL - Shadow Mapping - Z-Fighting



OpenGL - Shadow Mapping - Z-Fighting

- Problem: *Tiefentest in OpenGL schlägt aufgrund der geringen Auflösung und des logarithmischen Verhaltens des Tiefenbuffers fehl.*

- Lösung: *Bias Matrix anpassen.*

$$[Bias] = \begin{pmatrix} 0.5 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.499 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

OpenGL - Fehler abfangen/auswerten

- OpenGL generiert Laufzeitfehler
  - schnelleres Debugging
  - manuelles Abfangen nötig
  - Beispiel: **glPushMatrix**
    - kann GL_STACK_OVERFLOW, GL_STACK_UNDERFLOW und GL_INVALID_OPERATION generieren (Siehe Referenz 2.1)
- Fehlerprüfung sollte aus Performancegründen im Debug Mode durchgeführt und im Release Mode ignoriert werden

OpenGL - Fehler abfangen/auswerten

```
bool
GLErrorManager::checkAction( std::string const& action ) {
    unsigned err = glGetError();
    bool ret = true;
    switch(err) {
        case GL_NO_ERROR:
            break;
        case GL_INVALID_VALUE:
            std::cout << action << " (GL_INVALID_VALUE): ";
            std::cout << "A numeric argument is out of range. ";
            std::cout << "The offending command is ignored and has ";
            std::cout << "no other side effect than to set the error flag.";
            ret = false;
            break;
        case GL_INVALID_ENUM:
            std::cout << action << " (GL_INVALID_ENUM): An unacceptable";
            std::cout << " value is specified for an enumerated argument. ";
            std::cout << "The offending command is ignored and has no other ";
            std::cout << "side effect than to set the error flag.";
            ret = false;
            break;
        case GL_INVALID_OPERATION:
            // ...
        case GL_STACK_OVERFLOW:
            // ...
        case GL_STACK_UNDERFLOW:
            // ...
        case GL_OUT_OF_MEMORY:
            // ...
        case GL_TABLE_TOO_LARGE:
            // ...
        default:
            break;
    }
    return ret;
}
```

OpenGL - Fehler abfangen/auswerten

- _DEBUG ist in MVSC++ nur vordefiniert, wenn im *DEBUG Mode* kompiliert wird -> keine Fehlerprüfung im *RELEASE Mode*

```
        glPushMatrix();
#ifdef _DEBUG
        GLErrorManager::checkAction("someClass::someFunction(): glPushMatrix()");
#endif
```

Ressources - Linkliste

Shadow Mapping

http://www.cs.uiowa.edu/~cwyman/classes/common/gfxHandouts/shadowMapSteps.pdf
http://www.thomasannen.com/pub/bsc.pdf
http://www.paulsprojects.net/tutorials/smt/smt.html
http://www.opengl.org/wiki/Shadow_Mapping_without_shaders (Erweiterung zum letzten Link)

Shadow Volumes

http://www.informatik.uni-oldenburg.de/~brunhrn/shadows1.pdf
http://www.gamedev.net/reference/articles/article1873.asp
http://developer.nvidia.com/object/fast_shadow_volumes.html

FBO

http://www.gamedev.net/reference/articles/article2333.asp
http://www.gamedev.net/reference/articles/article2331.asp
http://wiki.delphigl.com/index.php/Tutorial_Framebufferobject
http://www.songho.ca/opengl/gl_fbo.html
http://www.opengl.org/registry/specs/EXT/framebuffer_object.txt <- offizielle Spec von OpenGL