VEKTORIA-MANUAL
HIERARCHITEKTUR-MODELLIERUNG

Prof. Dr. Tobias Breiner

vektoria

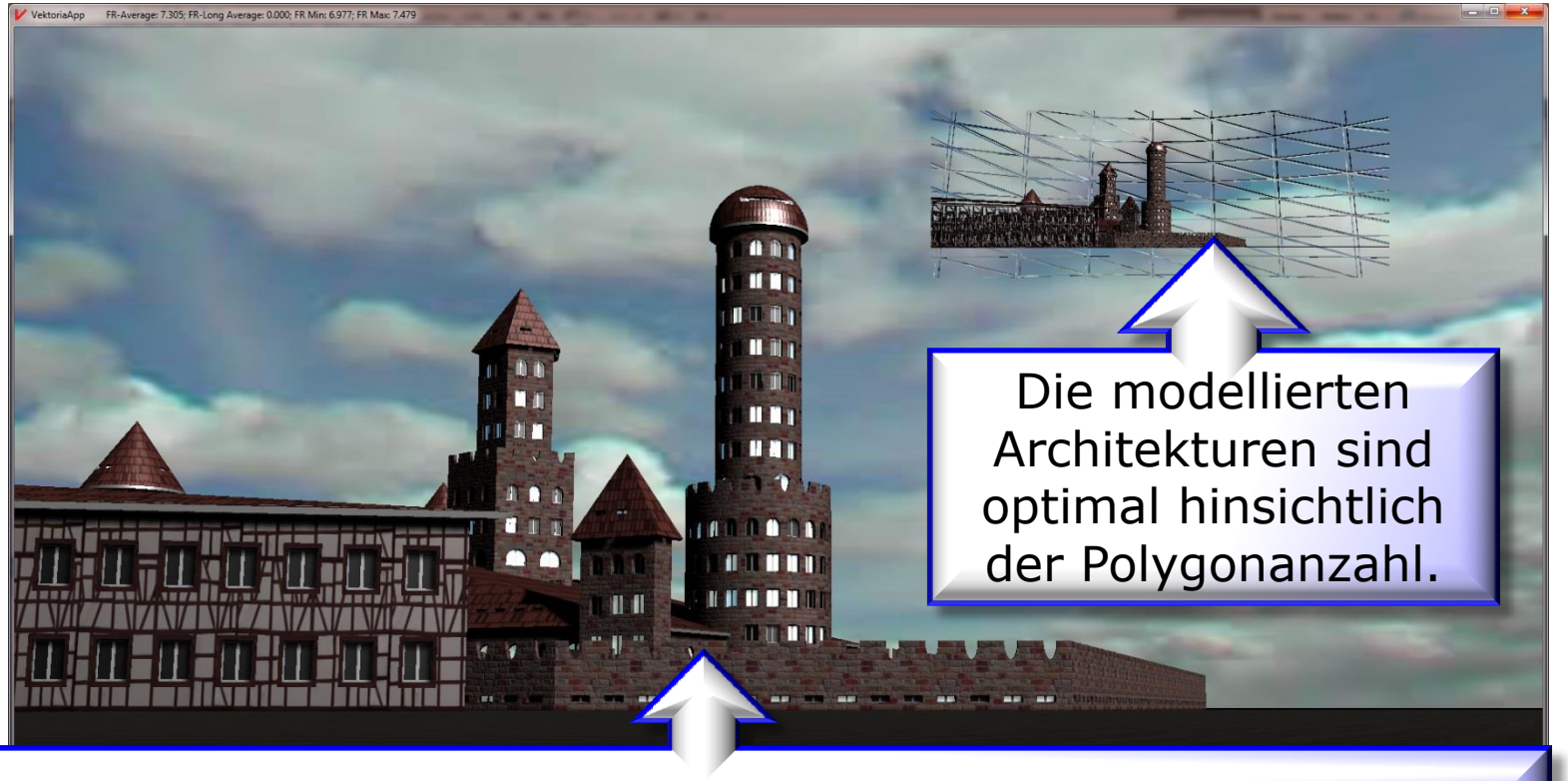
www.vektoria-engine.com

Inhalt

Was ist Hierarchitektur-Modellierung?

- Vektoria bietet als einziger Szenegraf bzw. Engine die sogenannte Hierarchitektur-Modellierung an.
- Hierarchitektur-Modellierung ist eine neue Methode zur einfachen und schnellen Erstellung von einzelnen Gebäuden, Gebäudekomplexen und anderen Architektur-strukturen mittels prozeduraler Objekthierarchiebildung.
- Mittels Hierarchitektur-Modellierung lassen sich aber auch viele andere technische Gegenstände schnell erstellen.
- Hierarchitektur-Modellierung kann beliebig mit Barr-Modellierung sowie Rippleing und Waving kombiniert werden, so dass eine Vielzahl von technischen Körpern erzeugbar ist.
- Hierarchitektur-Modellierung habe ich 1997 entwickelt und 2012 bis 2013 erstmals vollständig implementiert.

Was ist Hierarchitektur-Modellierung?



Komplexe Gebäude sind mit Hierarchitektur-Modellierung sehr schnell modellierbar (z.B. diese Burg binnen 23 Minuten).

HIERARCHITEKTURMOD.

2

3

4

5

Freischaltung

Hierarchitektur wird nur in einer sehr abgespeckten Version vom Basispaket von Vektoria angeboten.
Das Erstellen größerer Stadtstrukturen ist damit umöglich.

Falls Sie Interesse an den erweiterten Features haben, müssen Sie diese erst freischalten lassen.

Diesbezügliche Informationen unter: info@3D-Generation.de

3 Hauptobjekte der Hierarchitektur

Walls
(CGeoWall)

stellen einzelne Wandstrukturen dar. In sie können beliebig viele Fenster „eingestanzt“ werden. Auch Mauern, Böden, Dachflächen, usw. werden hier im erweiterten Sinne als Walls angesehen.

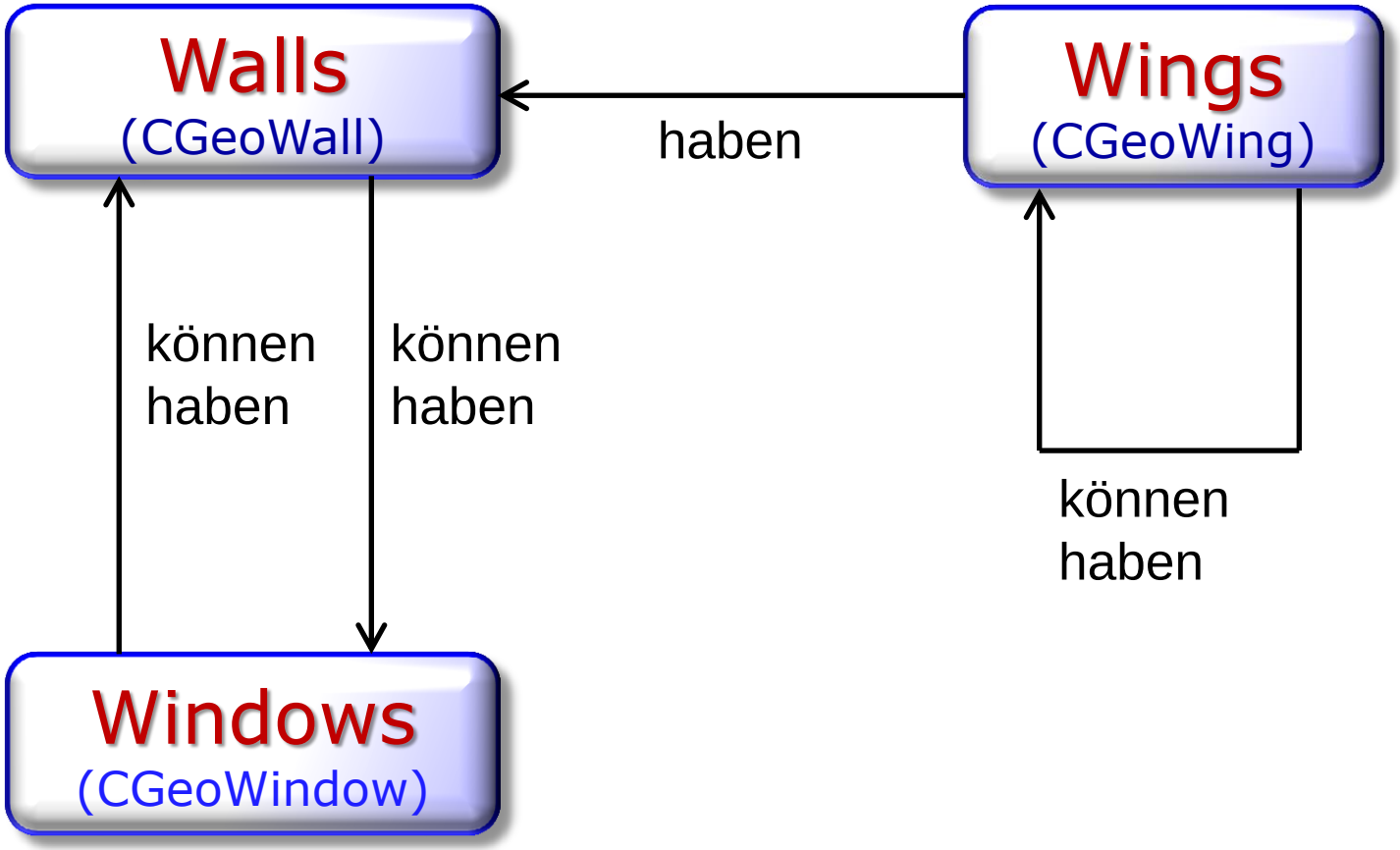
Windows
(CGeoWindow)

stellen Löcher oder Fensterdurchbrüche dar, die in Walls geschlagen werden. In diese Fenster können wiederum Unterwände eingesetzt werden.

Wings
(CGeoWing)

stellen beliebige Gebäudeflügel im erweiterten Sinne dar, z.B. ein Turm, Hauptbau, Anbau, Hof oder Erker. Die Flügel bestehen aus verschiedenen Walls und können beliebig viele Unterwings enthalten, so dass ein Gebäudekomplex entsteht.

Beziehung der WWW-Objekte



Ableitungshierarchie

Reihenfolge beachten! (1/2)

Bei Hierarchitektur-Modellierung muss die Befehlsreihenfolge strikt beachtet werden (siehe nächste Folie)!

Der Grund dafür ist einleuchtend:

- Der Aufbau der Geometrien kann nicht vollzogen werden, wenn nicht alle diesbezüglichen Parameter bekannt sind.
- Die Position der Vertices ist darüber hinaus abhängig von den via Add-Befehlen angehängten Eltern- und Kindobjekten.
- Die eventuelle Anwendung weiterer Methoden (Subdivition, Barr-Modellierung, Beschleunigung, etc.) kann erst dann erfolgen, wenn es schon irgendwelche Geometrien gibt, die man modifizieren kann.

Reihenfolge (2/2)

1. Verknüpfen der Objekte (Add-Methoden)
2. Parameter-Setting der Windows
3. Initialisierung der Windows
4. Parametersetting der Walls und Wings
5. Initalisierung der Walls und Wings
6. ev. Anwendung weiterer Modellierungen

Kapitel 2

1

WALLS

3

4

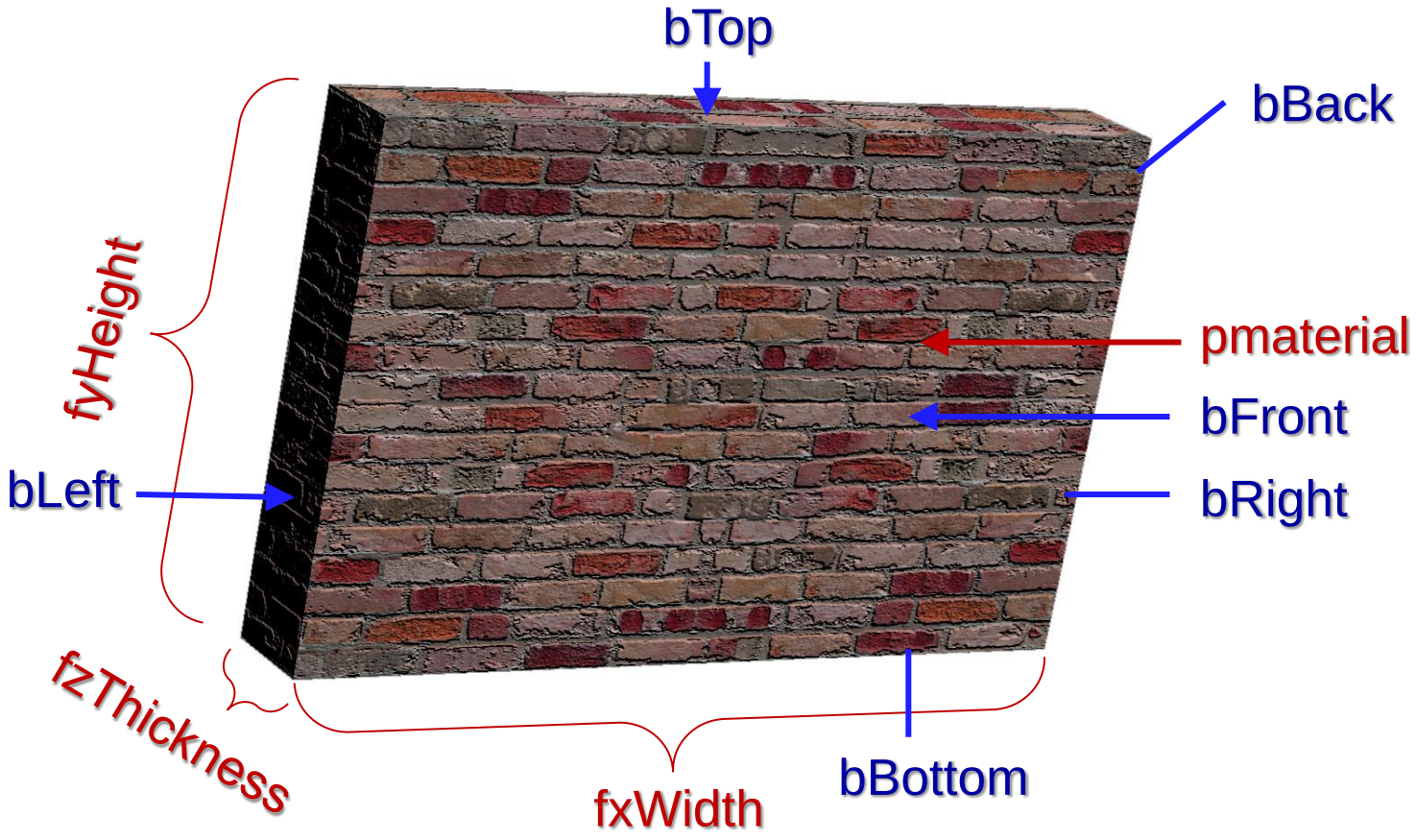
5

Was ist eine Wall?

Eine Wall wird durch die Klassse CGeoWall repräsentiert, die von CTriangleList und damit widerum von CGeo und CNode abgeleitet ist.
Eine Wall stellt eine einzelne Wandstruktur dar. Sie kann gerade, geneigt und gebogen sein. Auch kann sie seitliche Gehrungen und Giebel enthalten. In eine Wall können beliebig viele Fenster „eingestanzt“ werden.

Auch Mauern, Böden, Dachflächen, usw. werden hier im erweiterten Sinne als Walls angesehen, so dass eine Wall eine Obermenge von Wänden ist.

Begriffsbestimmung der Parameter (1/2)



Initialisierung einer geraden Wall (2/2)

```
void Init(
        float fxWidth,
        float fyHeight,
        float fzThickness,
        CMaterial * pmaterial,
        bool bLeft = true,
        bool bRight = true,
        bool bBottom = true,
        bool bTop = true,
        bool bFront = true,
        bool bBack = true);
```

Ausmaße

Materialzeiger

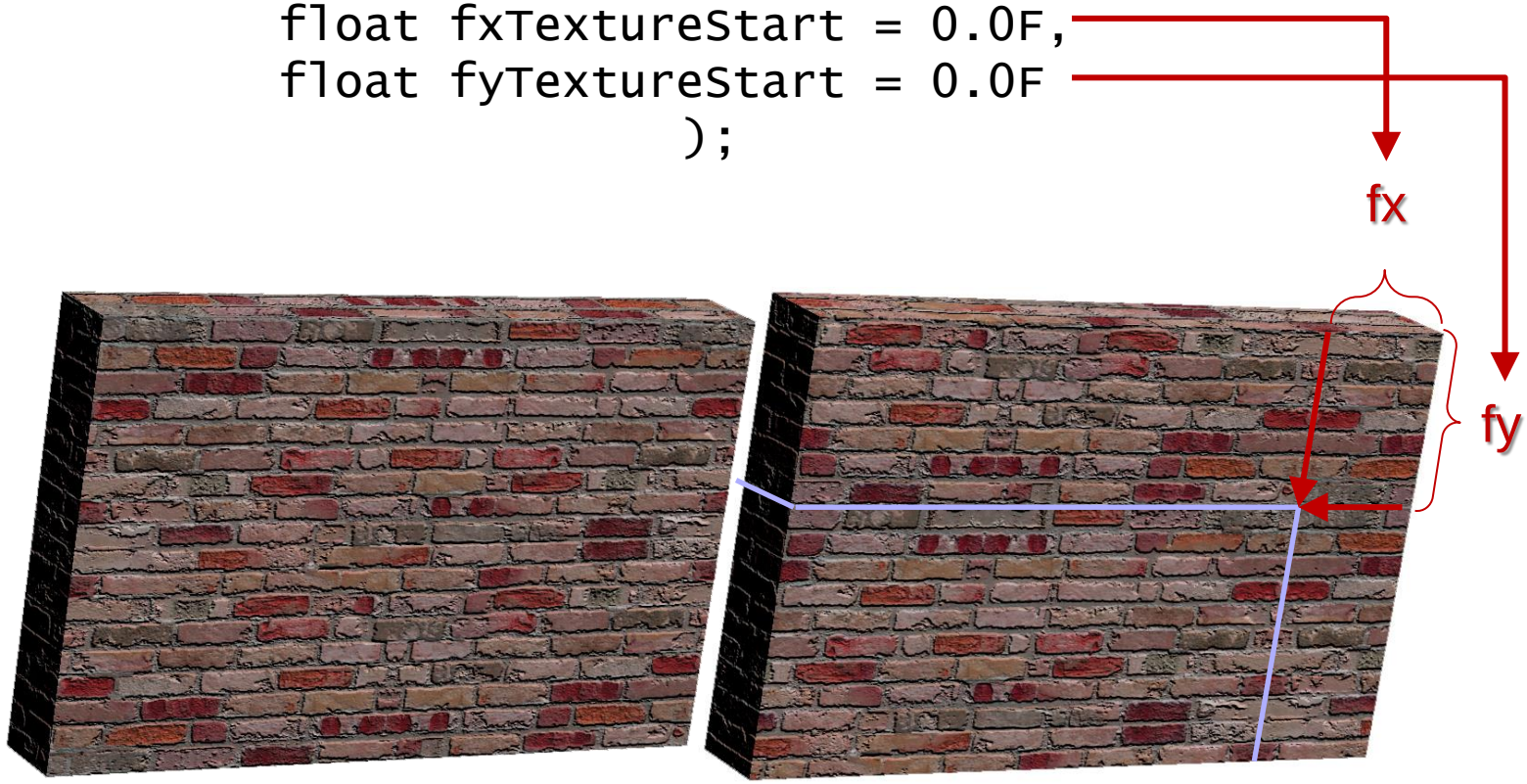
Seitenanzeigeflags

Initialisiert die Mauer. Der Parameter fxWidth ist die Länge, fyHeight die Höhe, fzThickness die Dicke, pmaterial das Material der Mauer. Mit den boolschen Flags lassen sich die entsprechenden Seiten an- und ausschalten.

CGeoWall

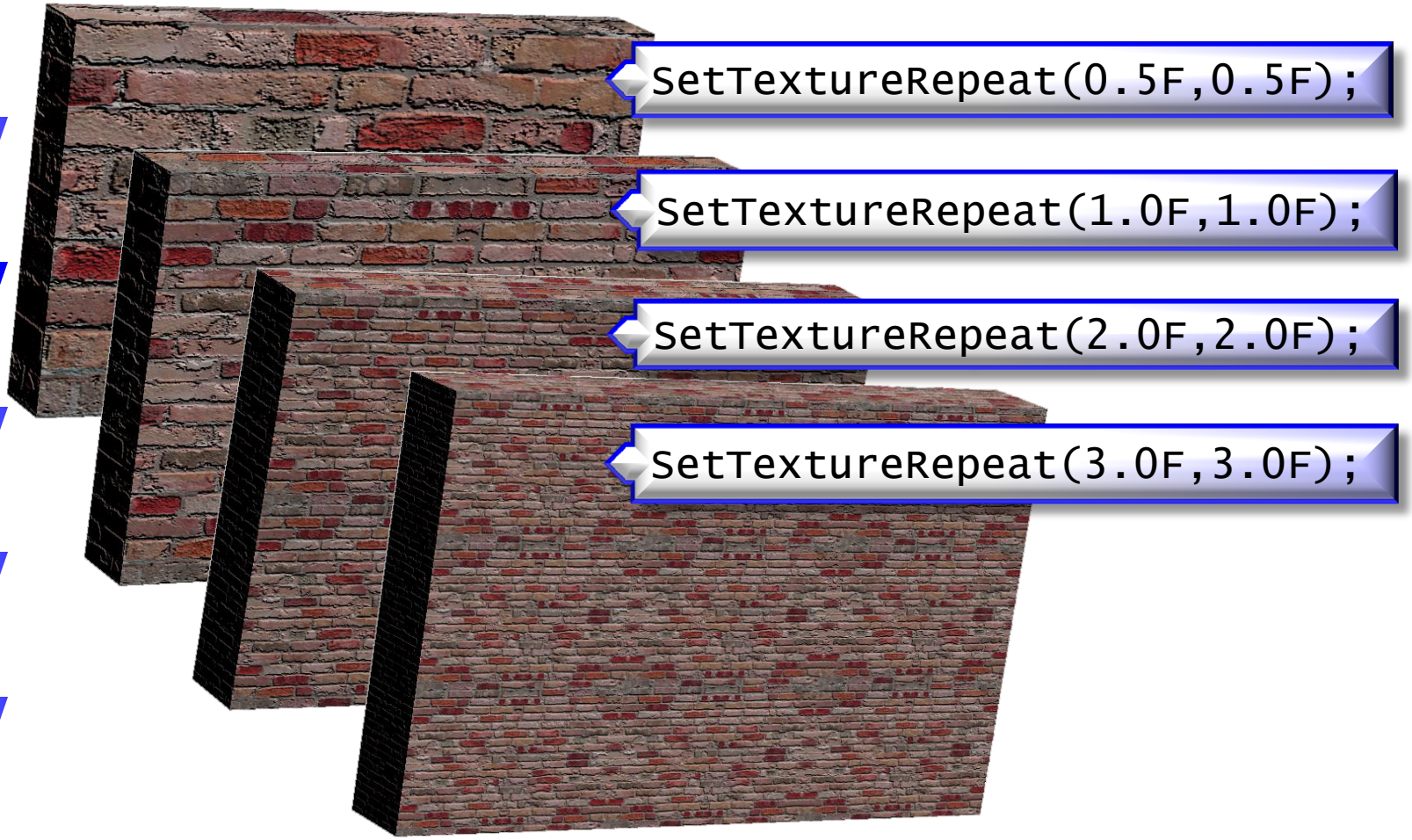
SetTextureStart

```
void SetTextureStart(
        float fxTextureStart = 0.0F,
        float fyTextureStart = 0.0F
                        );
```



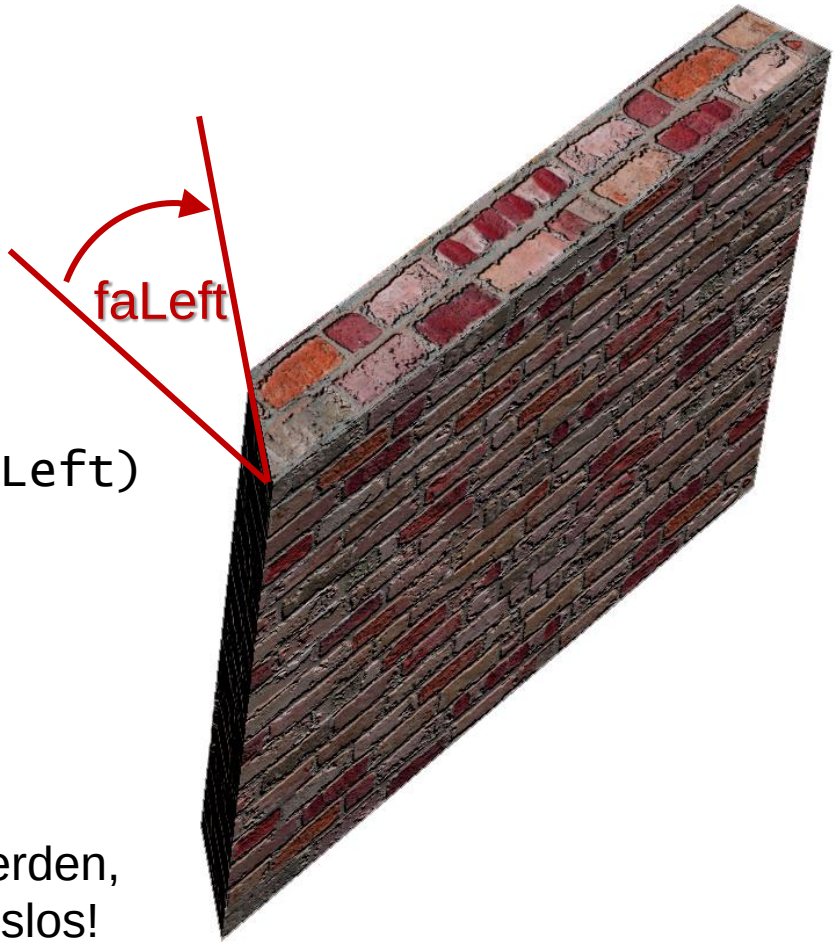
CGeoWall

SetTextureRepeat



1
2
3
4
5

SetMiterLeft




```
SetMiterLeft(float faLeft)
```

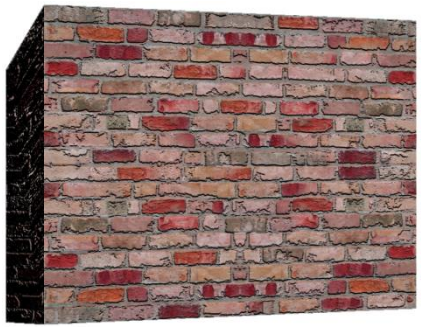
Zur Erinnerung:

Hierarchietektur-Setting-Methoden sollten *vor* der Initalisierung aufgerufen werden, ansonsten sind sie wirkungslos!

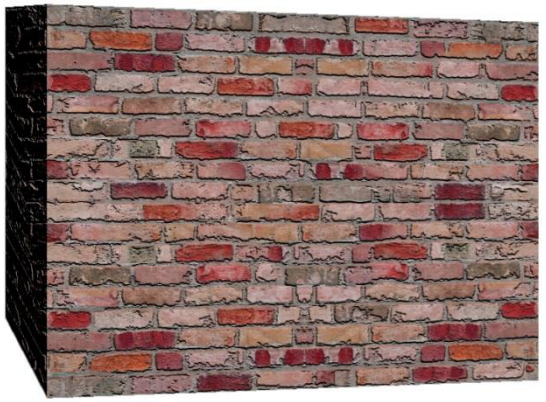
SetMiter-Methoden

SetMiterLeft(float faLeft)

SetMiterRight(float faRight)



SetMiterTop(float faTop)



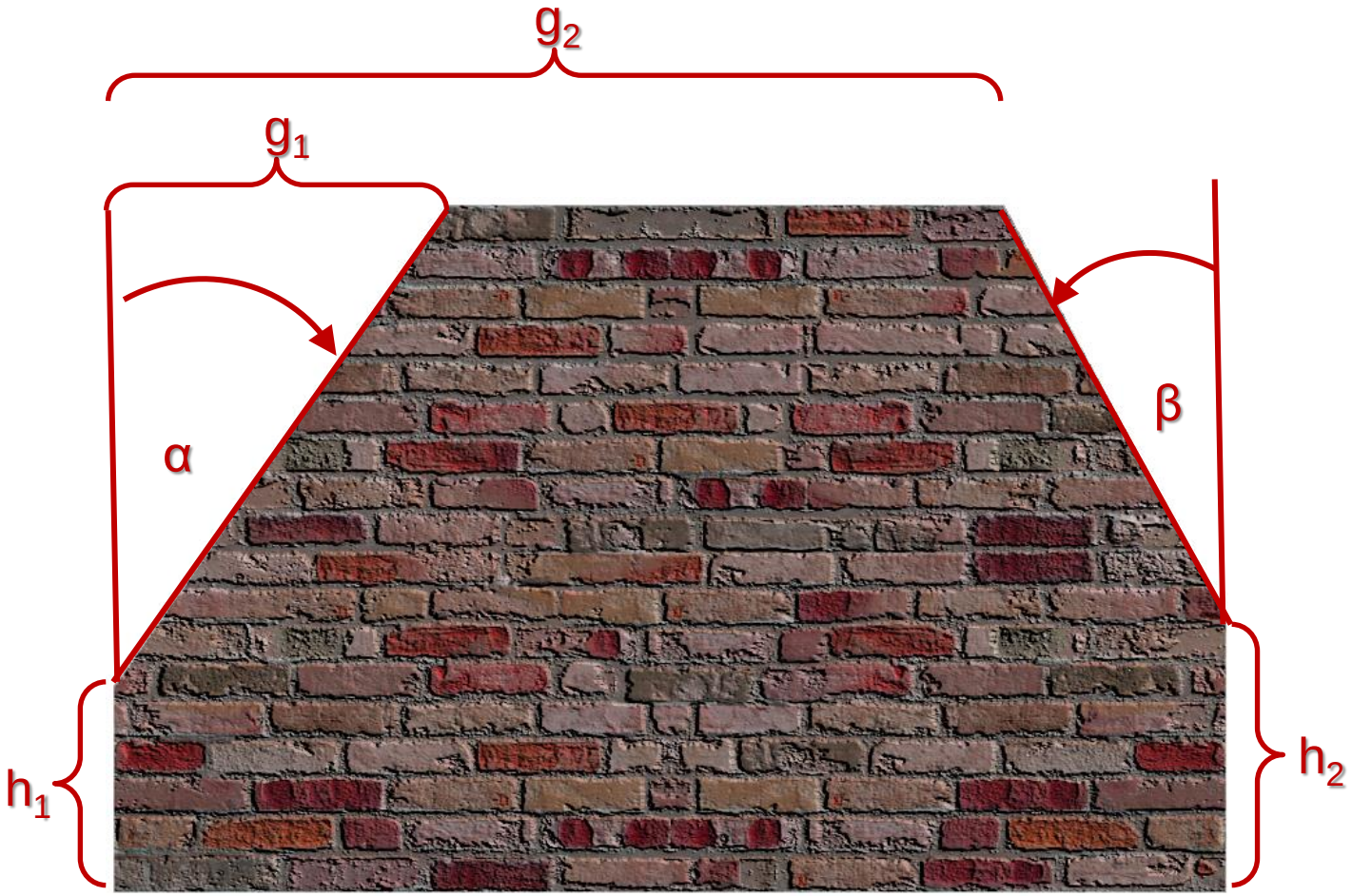
SetMiterBottom(float faBottom)

Übung „Drei Mauern“ (ohne Lösung)

1. Erzeugen Sie drei Mauern und stellen Sie diese hinsichtlich des Grundrisses in Form eines gleichseitigen Dreieicks auf!

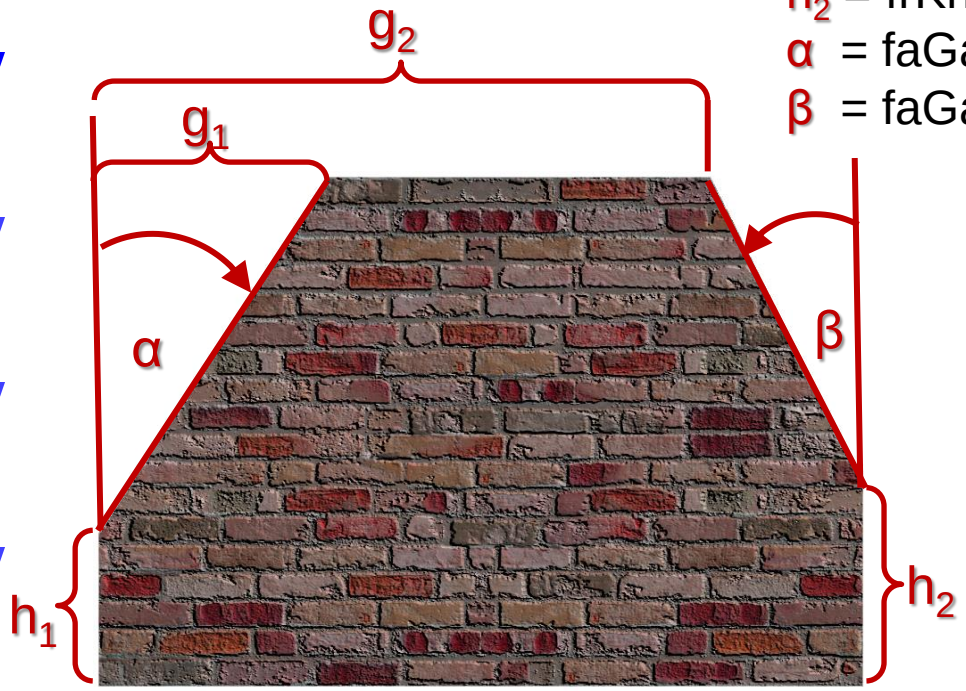
4. Freiwillige Übung für die schnellen Nerds:
   Erzeugen Sie ein Mauergeflecht in Form eines Pentagrammes!

Giebelpunkte und Kniestockhöhe (1/3)



Giebelpunkte und Kniestockhöhe (2/3)

g_1 = frCuspLeft*fxWidth
g_2 = frCuspRight*fxWidth
h_1 = frKneeWallHeightLeft*fxHeight
h_2 = frKneeWallHeightRight*fxHeight
α = faGableLeft
β = faGableRight



CGeoWall

Giebelpunkte und Kniestockhöhe (3/3)

1

```
void SetGable
  (float frCuspLeft,
   float frCuspRight);
```

Setzt Giebelpunkte links und rechts auf unterschiedliche Positionen

2

```
void SetGable
  (float frCusp);
```

Setzt Giebelpunkte links und rechts auf gemeinsame Position (Spitzgiebel)

3

4

```
void SetGableLeft
  (float frCuspLeft);
```

Setzt nur linken Giebelpunkt

5

```
void SetGableRight
  (float frCuspRight);
```

Setzt nur rechten Giebelpunkt

Giebelpunkte und Kniestockhöhe (3/3)

1
```
void SetKneeWallHeight
  (float frKneeWallHeightLeft,
   float frKneeWallHeightRight);
```
Setzt Kniestock links und rechts auf unterschied-liche Höhen

2
```
void SetKneeWallHeight
  (float frKneeWallHeight);
```
Setzt Kniestock links und rechts auf gleiche Höhe

3
```
void SetKneeWallHeightLeft
  (float frKneeWallHeightLeft);
```
Setzt nur linke Kniestockhöhe

4
```
void SetKneeWallHeightRight
  (float frKneeWallHeightRight);
```
Setzt nur rechte Kniestockhöhe

5

Horizontal gebogene Wände

```
void SetRoundingX(
        float fa,
        float fLengthMax = 0.2f);
```

Biegungswinkel im Bogenmaß

Minimalkantenlänge in X-Richtung (je kleiner, desto mehr Polygone)

Zur Erinnerung:

Hierarchietektur-Setting-Methoden sollten *vor* der Initalisierung aufgerufen werden, ansonsten sind sie wirkungslos!

Vertikal gebogene Wände

```
void SetRoundingY(
        float fa,
        float fLengthMax = 0.2f);
```

Biegungswinkel im Bogenmaß

Minimalkantenlänge in Y-Richtung (je kleiner, desto mehr Polygone)

Kombinationen aus SetRoundingX und -Y

Für geschwungene Wände kann man SetRoundingX und SetRoundingY miteinander kombinieren.

Achtung!

Werte über PI bei den kombinierten Winkelangaben können zu Artefakten führen, da es dann naturgemäß zu einer partiellen Invertierung der Vertexreihenfolge kommt!

Übung „Tunnel“

Erzeugen Sie einen Tunnel!

Freiwillige Übung für die schnellen Nerds:
Erzeugen Sie einen gebogenen Tunnel!

Lösung „Tunnel“ (1/4)

1
2
3
4
5

```
void CGame::Init(HWND hwnd, HWND hwndDX,
                                    RECT rectWnd)
{
  …
  …
  // Biegung auf π (=180°) setzen
  // und kleinen Minimalkantenabstand wählen (0,02):
  m_zgWall.SetRoundingY(PI,0.02F);
  // Wand initialisieren (mit
  // X-Länge = 2 Einheiten, Y-Höhe = 5 Einheiten,
  // Z-Breite = 0,2 Einheiten):
  m_zgWall.Init(2.0F,5.0F, 0.2F,&m_zm);
  …
}
```

Lösungsausgabe „Tunnel“(2/4)

Nerdlösung „Tunnel“ (3/4)


```
void CGame::Init(HWND hwnd, HWND hwndDX,
                                    RECT rectWnd)
{
  …
  …
  // einfachen Tunnel erzeugen
  // (Siehe vorherige Lösung):
  m_zgWall.SetRoundingY(PI,0.02F);
  m_zgWall.Init(2.0F,5.0F, 0.2F,&m_zm);
  // Für genügend Kanten zum Verbiegen sorgen:
  m_zgWall.Subdivide(0.02F);
  // Mit der Barr-Methode „BendX“ einen U-Turn
  // erzeugen (½π*2 = π  entsprechend 180°):
  m_zgWall.BendX(3.0F,HALFPI);
  …
}
```

1
2
3
4
5

Nerdlösungsausgabe „Tunnel“ (4/4)

Froschperspektive:

Vogelperspektive:

Initialisierung einer geraden Wall (2/2)

```
void Init(
        float fxWidth,
        float fyHeight,
        float fzThickness,
        CMaterial * pmaterial,
        bool bLeft = true,
        bool bRight = true,
        bool bBottom = true,
        bool bTop = true,
        bool bFront = true,
        bool bBack = true);
```

Ausmaße

Materialzeiger

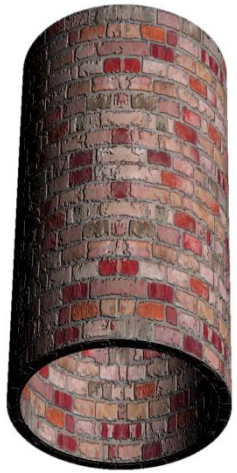
Seitenanzeigeflags

Initialisiert die Mauer. Der Parameter fxWidth ist die Länge, fyHeight die Höhe, fzThickness die Dicke, pmaterial das Material der Mauer. Mit den boolschen Flags lassen sich die entsprechenden Seiten an- und ausschalten.

Initialisierung einer Zylinderwand

```
void InitTube(
        float fRadius,
        float fThickness,
        CMaterial * pmaterial = 0,
        float fEdgeLengthMin = 0.5F,
        bool bBottom = true,
        bool bBack = true);
```

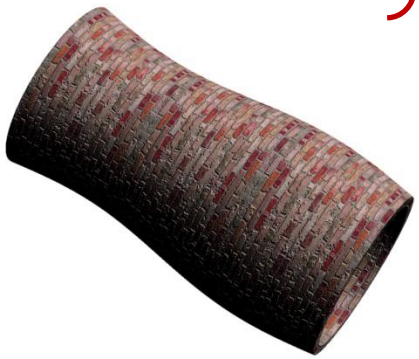
- fRadius → Innenradius
- fThickness → Wanddicke
- pmaterial → Materialzeiger
- fEdgeLengthMin → Radiale Minimalkantenlänge (je kleiner, desto mehr Polygone)
- bBottom, bBack → Seitenanzeigeflags



Initial. zirkuläre Wellenwand

```
void InitTubeWave(
        float fRadius,
        float fHeight,
        float fThickness,
        float fAmplitude = 1.0F,
        float fWavelength = 1.0F,
        CMaterial * pmaterial = 0,
        float fEdgeLengthMin = 0.2F,
        bool bBottom = true,
        bool bBack = true);
```

- fRadius → Innenradius
- fHeight → Höhe der Spitze
- fThickness → Wanddicke
- fAmplitude → Amplitude
- fWavelength → Wellenlänge
- pmaterial → Materialzeiger
- fEdgeLengthMin → Radiale Minimalkantenlänge (je kleiner, desto mehr Polygone)
- bBottom, bBack → Seiten-anzeige-flags



Initialisierung Schnittstellenzylinder

```
void InitTubeInterface(
        float fRadiusBottom,
        float fRadiusTop,
        float fHeight,
        float fThickness,
        CMaterial * pmaterial = 0,
        float fEdgeLengthMin = 0.5F,
        bool bBottom = true,
        bool bBack = true);
```

- fRadiusBottom → Radius unten
- fRadiusTop → Radius oben
- fHeight → Höhe
- fThickness → Wanddicke
- pmaterial → Materialzeiger
- fEdgeLengthMin → Radiale Minimalkantenlänge (je kleiner, desto mehr Polygone)
- bBottom, bBack → Seitenanzeigeflags

Initialisierung Sinuszylinder

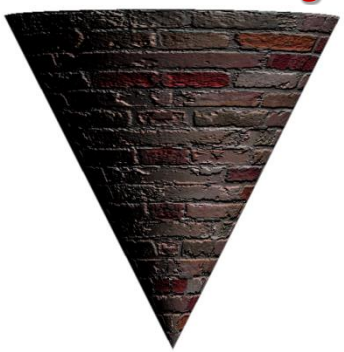
```
void InitTubeSine(
        float fRadiusBottom,
        float fRadiusTop,
        float fHeight
        float fThickness,
        CMaterial * pmaterial = 0,
        float fEdgeLengthMin = 0.5F,
        bool bBottom = true,
        bool bBack = true);
```

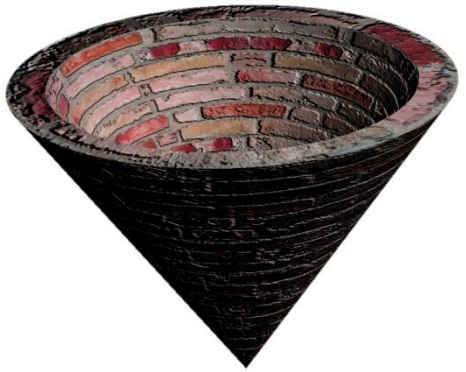
- fRadiusBottom → Radius unten
- fRadiusTop → Radius oben
- fHeight → Höhe
- fThickness → Wanddicke
- pmaterial → Materialzeiger
- fEdgeLengthMin → Radiale Minimalkantenlänge (je kleiner, desto mehr Polygone)
- bBottom, bBack → Seitenanzeigeflags

Initialisierung Runderkerkegel

```
void InitTubeOriel(
        float fRadius,
        float fHeight,
        float fThickness,
        CMaterial * pmaterial = 0,
        float fEdgeLengthMin = 0.5F,
        bool bBottom = true,
        bool bBack = true);
```

- fRadius → Radius oben
- fHeight → Höhe
- fThickness → Wanddicke
- pmaterial → Materialzeiger
- fEdgeLengthMin → Radiale Minimalkantenlänge (je kleiner, desto mehr Polygone)
- bBottom, bBack → Seitenanzeigeflags

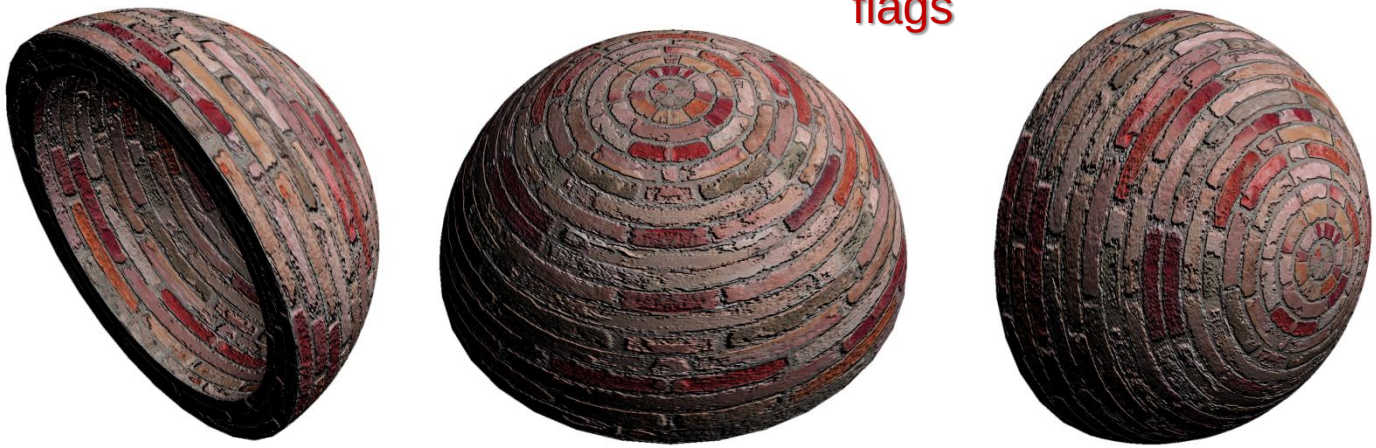




Initialisierung einer Kuppelwand

```
void InitDome(
        float fRadius,
        float fThickness,
        CMaterial * pmaterial = 0,
        float fEdgeLengthMin = 0.5F,
        bool bBottom = true,
        bool bBack = true);
```

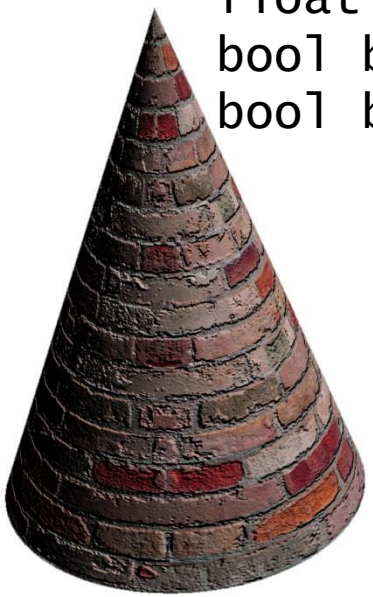
- fRadius → Innenradius
- fThickness → Wanddicke
- pmaterial → Materialzeiger
- fEdgeLengthMin → Minimalkantenlänge (je kleiner, desto mehr Polygone)
- bBottom, bBack → Seiten-Anzeige-flags



Initialisierung einer Runddachwand

```
void InitDomeCone(
        float fRadius,
        float fHeight,
        float fThickness,
        CMaterial * pmaterial = 0,
        float fEdgeLengthMin = 0.5F,
        bool bBottom = true,
        bool bBack = true);
```

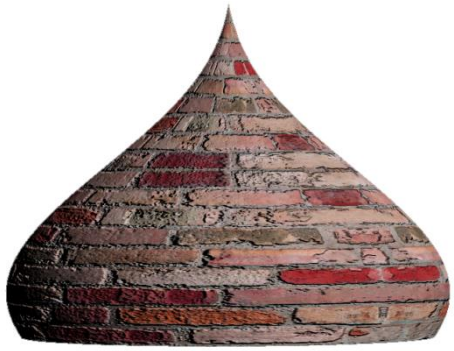
- fRadius → Innenradius
- fHeight → Dachhöhe
- fThickness → Wanddicke
- pmaterial → Materialzeiger
- fEdgeLengthMin → Minimalkantenlänge (je kleiner, desto mehr Polygone)
- bBottom, bBack → Seiten-Anzeige-flags



Initial. Zwiebeltürmchenwand

```
void InitDomeOnion(
        float fRadius,
        float fHeight,
        float fThickness,
        float fStrength = 0.7F,
        CMaterial * pmaterial = 0,
        float fEdgeLengthMin = 0.2F,
        bool bBottom = true,
        bool bBack = true);
```

- fRadius → Innenradius
- fHeight → Dachhöhe
- fThickness → Wanddicke
- fStrength → Krümmungsstärke
- pmaterial → Materialzeiger
- fEdgeLengthMin → Minimalkantenlänge (je kleiner, desto mehr Polygone)
- bBottom, bBack → Seitenanzeigeflags



Initial. Stachelwand

```
void InitDomeSpike(
        float fRadius,
        float fHeight,
        float fThickness,
        float fStrength = 0.7F,
        CMaterial * pmaterial = 0,
        fEdgeLengthMin = 0.2F,
        bBottom = true,
        bBack = true);
```

- fRadius → Innenradius
- fHeight → Wanddicke
- fThickness → Dachhöhe
- fStrength → Materialzeiger
- pmaterial → Minimalkantenlänge
- fEdgeLengthMin → (je kleiner, desto mehr Polygone)
- bBottom, bBack → Seiten-anzeige-flags

Adjust

Problem: Manchmal wollen wir verschiedene Walls aufeinander setzen, dabei kann es bei gebogenen Wänden und Rundeturmteilen sein, dass sie unterschiedliche Auflösungen besitzen, was zu kleinen Lücken zwischen den Walls führt. Um dies nachträglich zu verhindern, gibt es in der Klasse CGeoWall die Methode „Adjust".

```
void Adjust(
        CGeoWall & geowallTop,
        bool bAdjustMapping = true
            );
```

Justiert die oberen Vertices so, dass sie zur Wand geowallTop passen. Wenn bAdjustMapping auf true gesetzt wird, werden ebenfalls die UV-Koordinaten beider Teile angepasst.

Übung „Leuchtturm“ (ohne Lösung)

Erzeugen Sie einen Leuchtturm Ihrer Wahl!

1

2

3

4

5

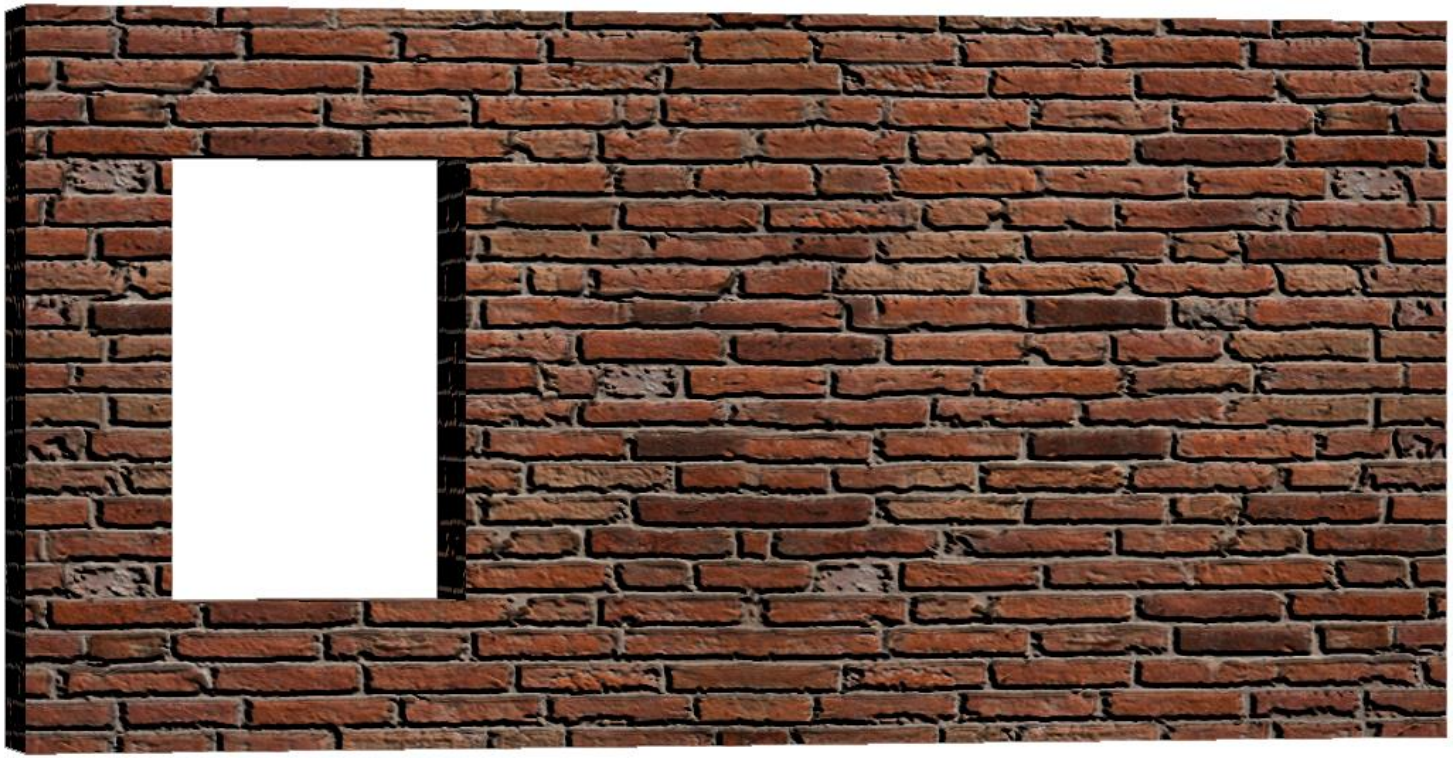
Freiwillige Übung für die schnellen Nerds:
Lassen Sie ausgehend von der Leuchtturmspitze ein Scheinwerferlicht rotieren!

Kapitel 3

Was ist ein Window?

1. Ein Window wird durch die Klassse CGeoWindow repräsentiert. Es stellt eine fensterartige Ausparung in einer einzelnen Wandstruktur dar. Diese Aussparungen können viele Formen annehmen.
2. Es können beliebig viele Windows in eine Wall „eingestanzt“ werden.
3. In diese Fenster können wiederum Unterwände eingesetzt werden, was eine große Komplexität der modellierten Strukturen erlaubt.

Fensterformen in CGeoWindow



Rect

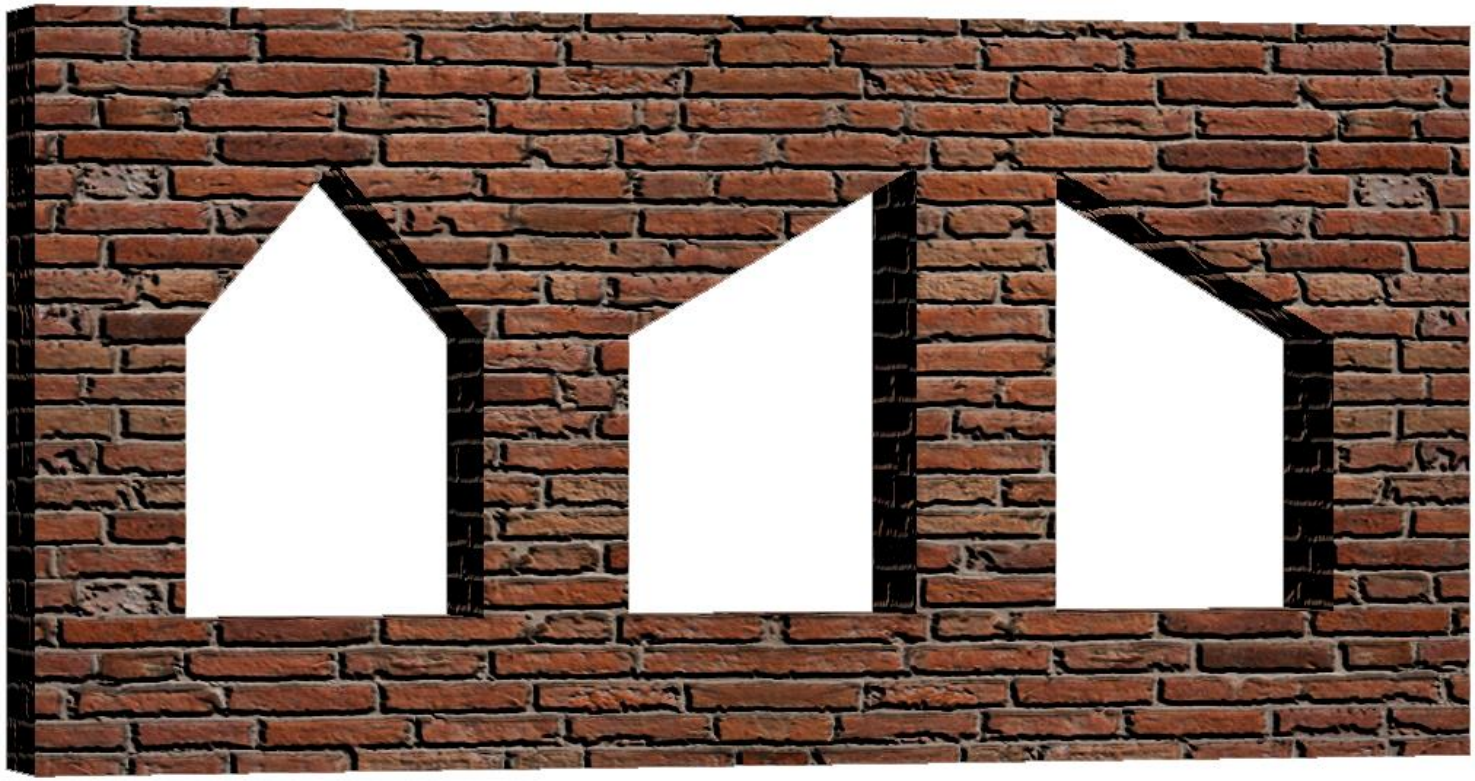
Fensterformen in CGeoWindow

Tri TriLeft TriRight

Fensterformen in CGeoWindow

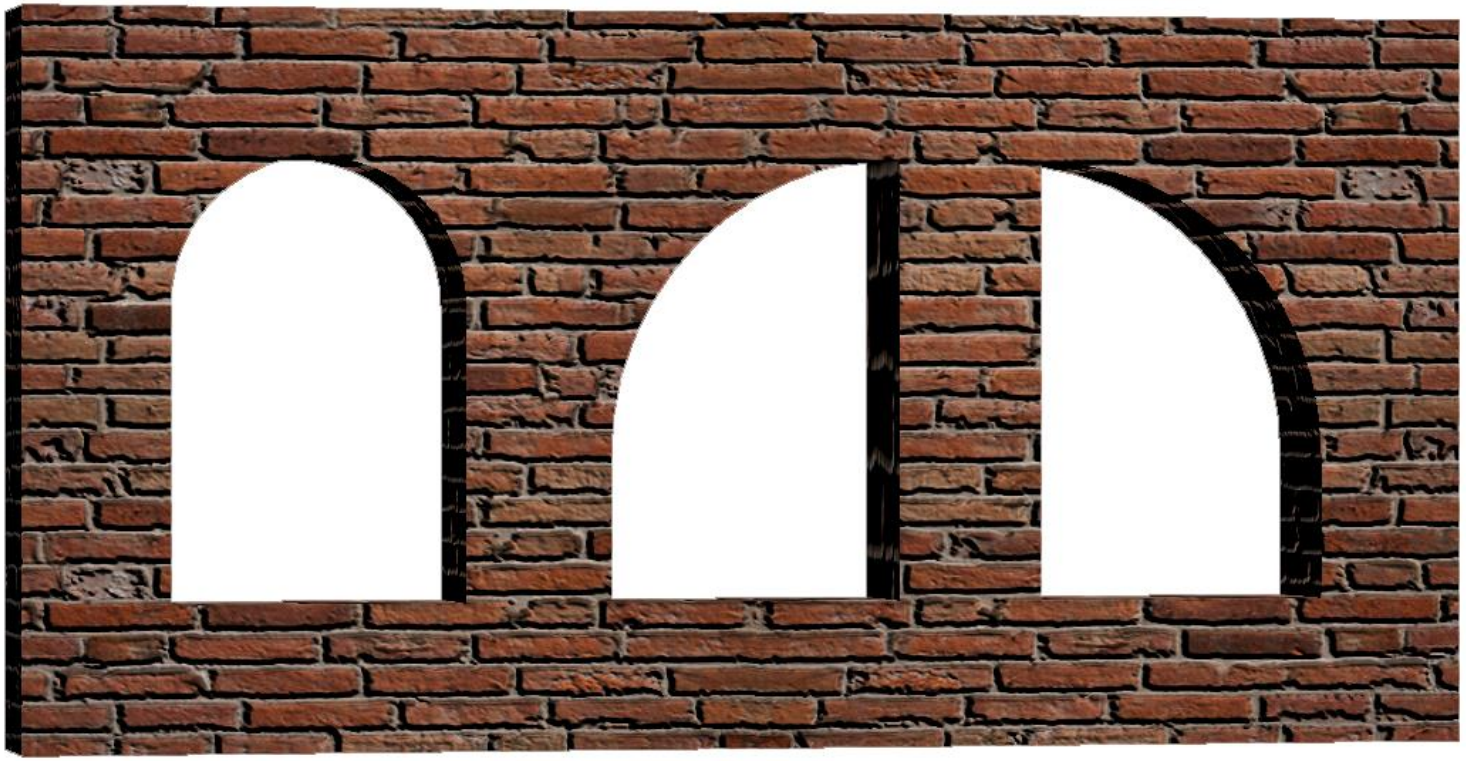
V VLeft VRight

Fensterformen in CGeoWindow



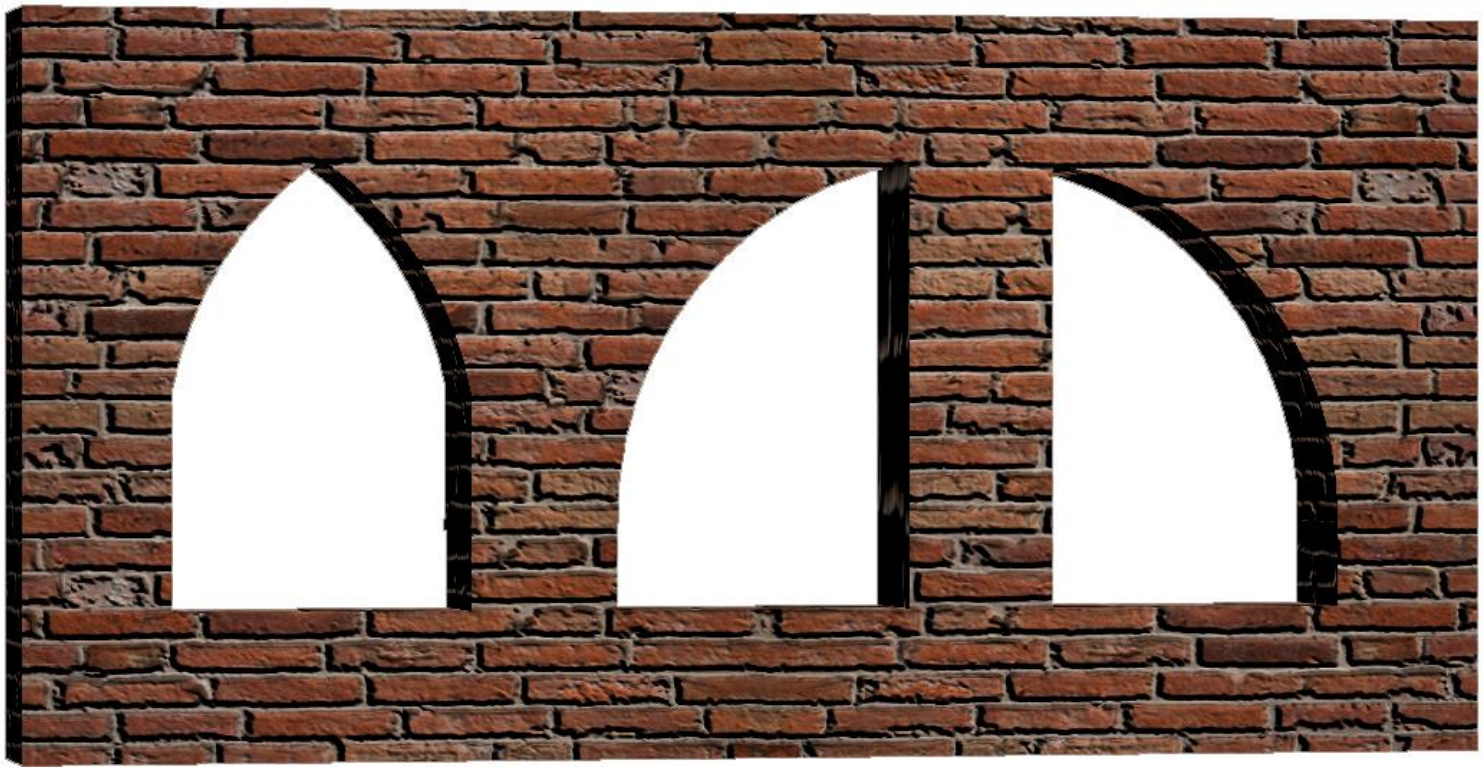
TriRect TriRectLeft TriRectRight

Fensterformen in CGeoWindow



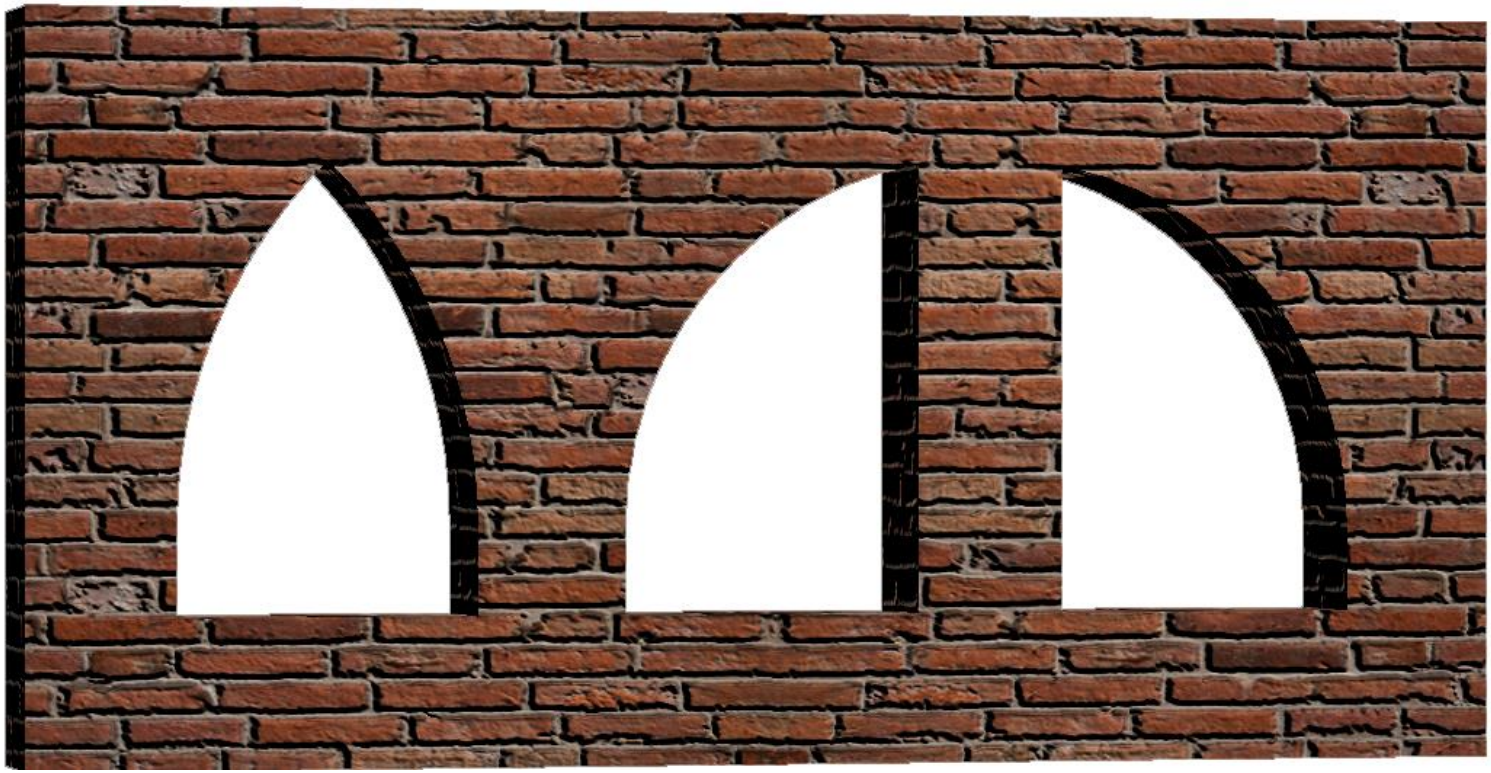
Roman RomanLeft RomanRight

Fensterformen in CGeoWindow



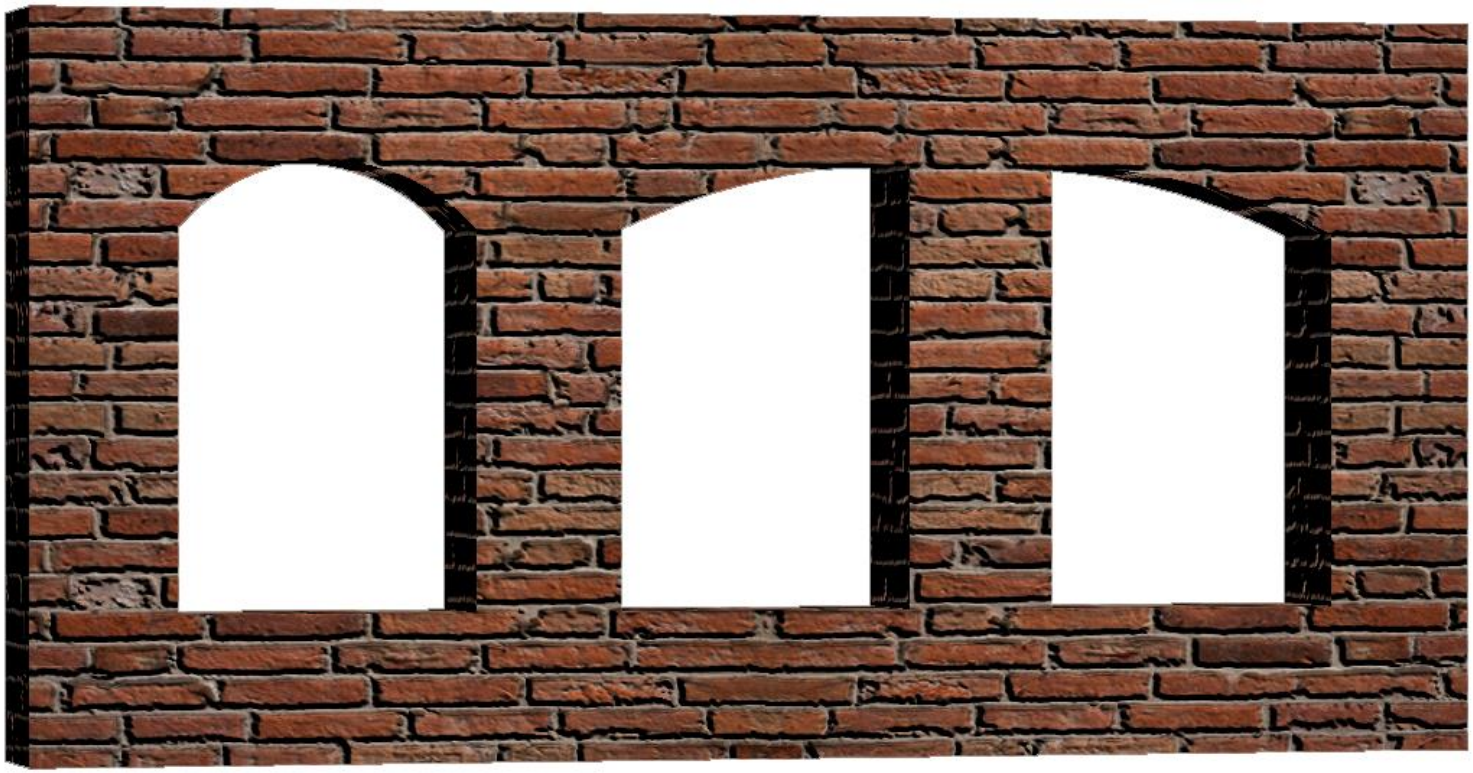
Arcade ArcadeLeft ArcadeRight

Fensterformen in CGeoWindow



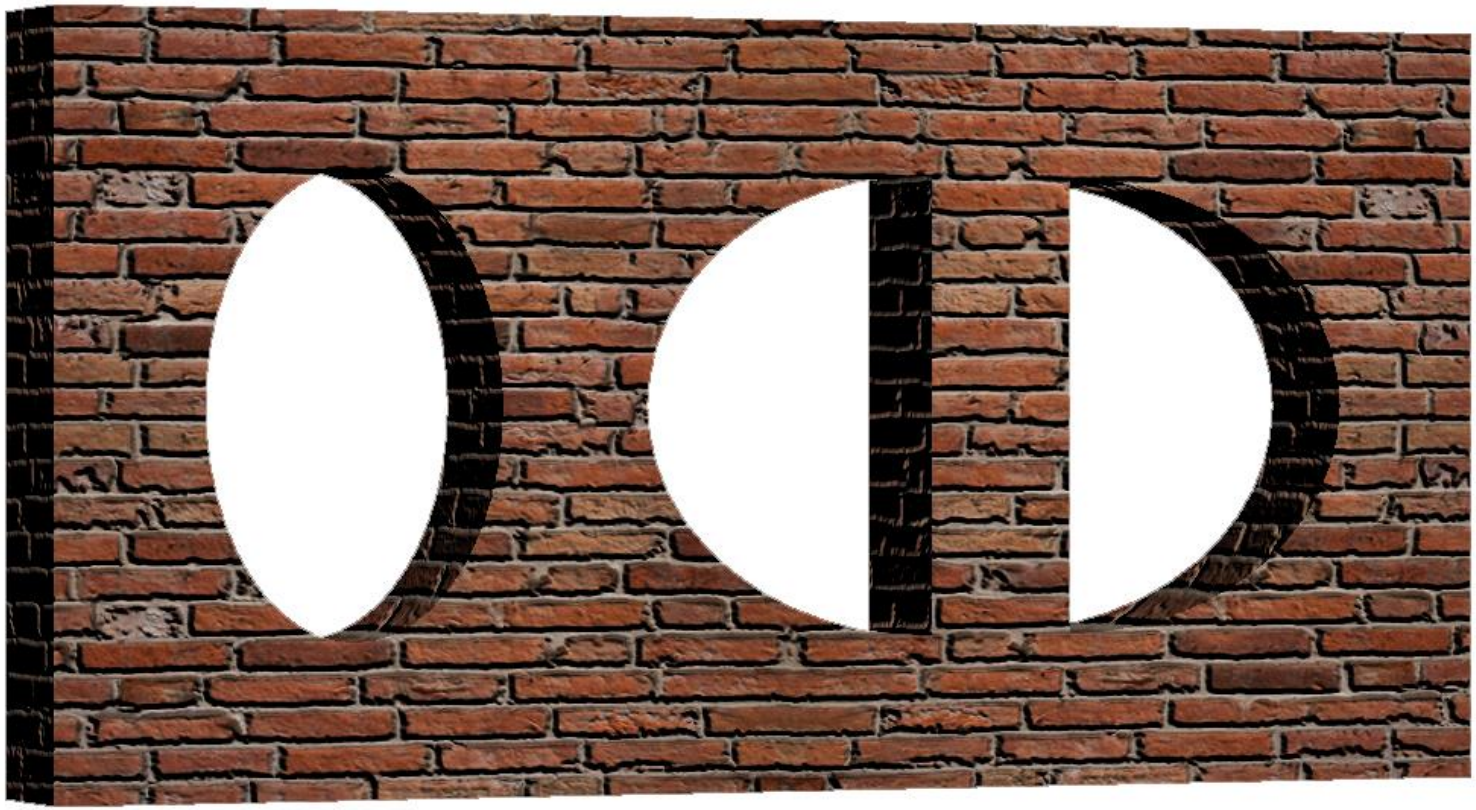
Gothic GothicLeft GothicRight

Fensterformen in CGeoWindow



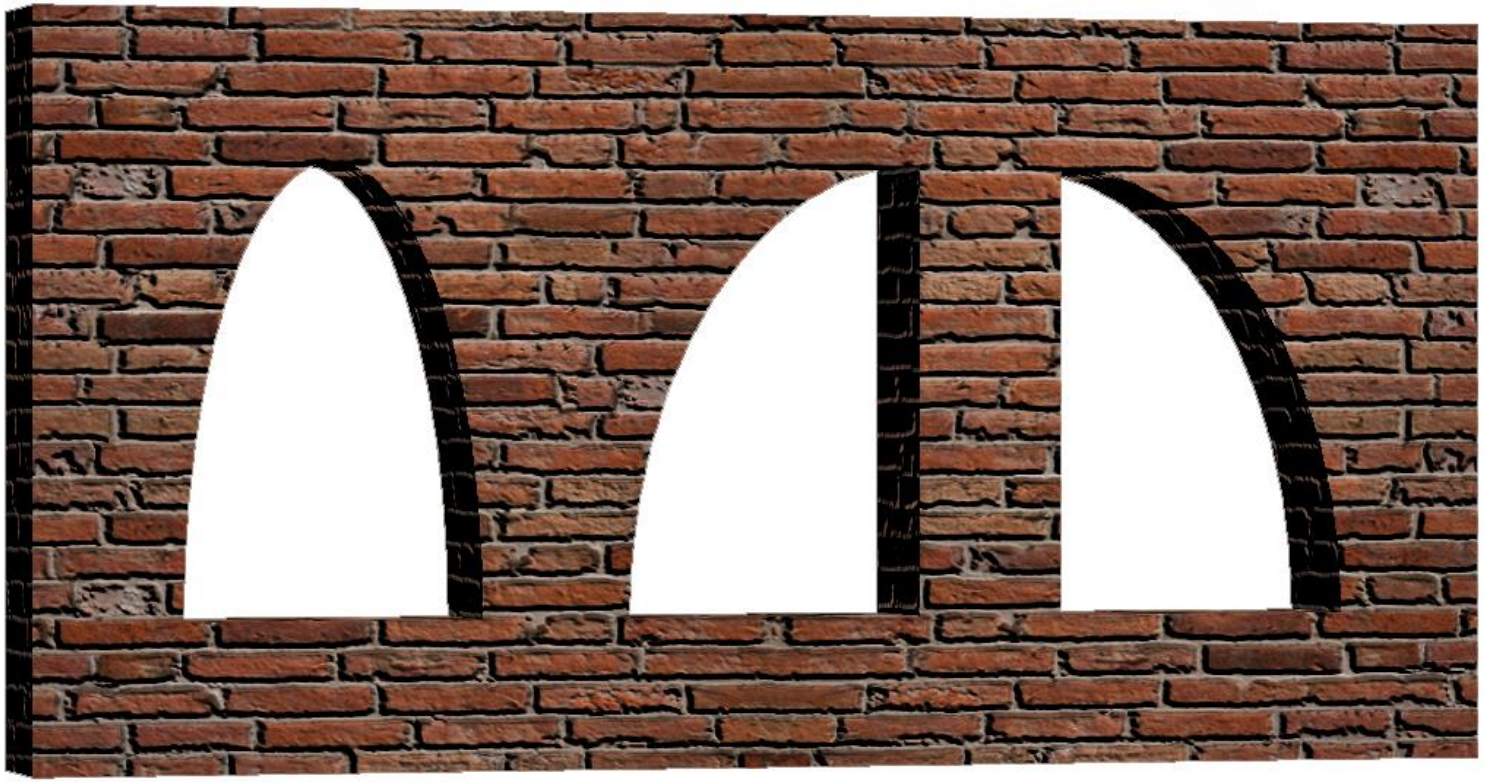
Eyebrow EyebrowLeft EyebrowRight

Fensterformen in CGeoWindow



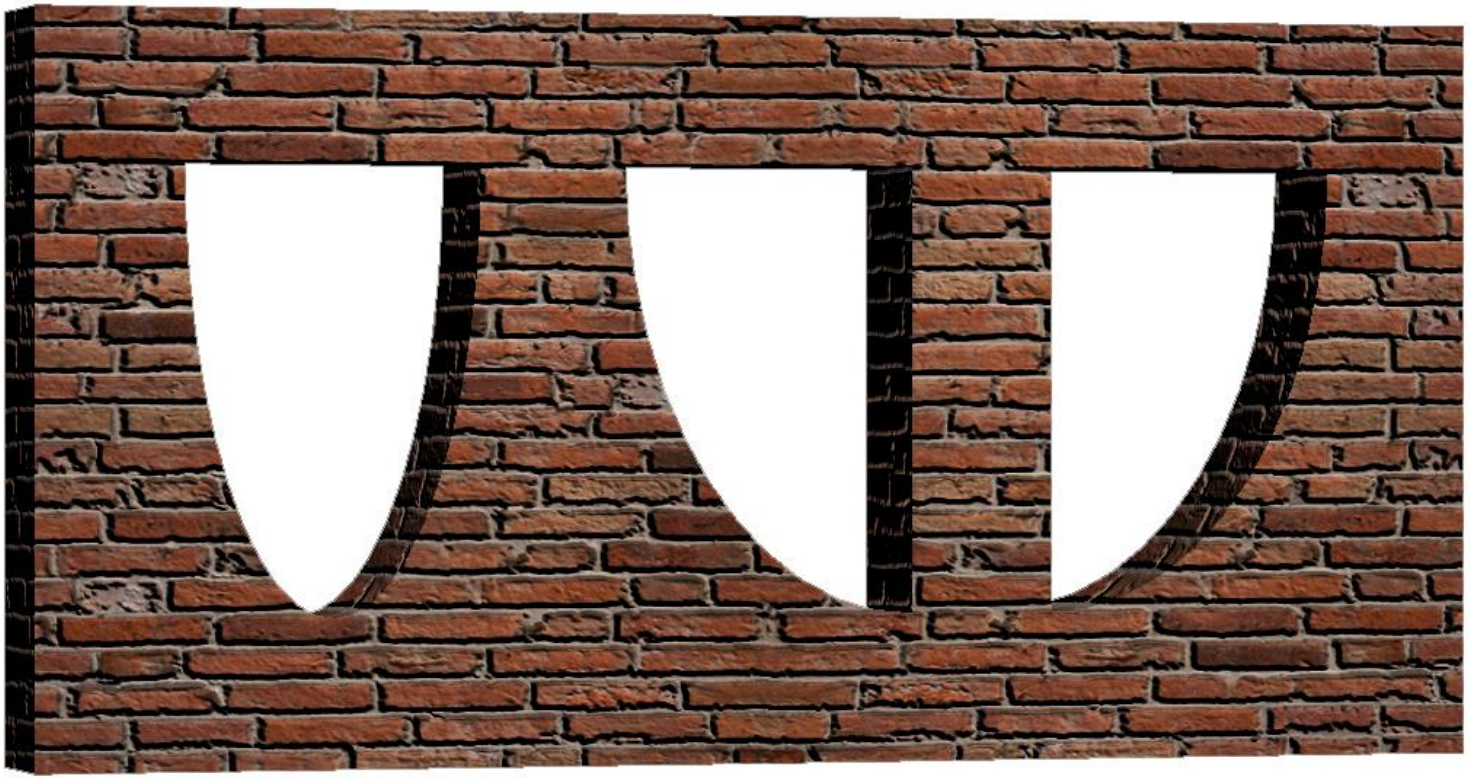
Oval OvalLeft OvalRight

Fensterformen in CGeoWindow



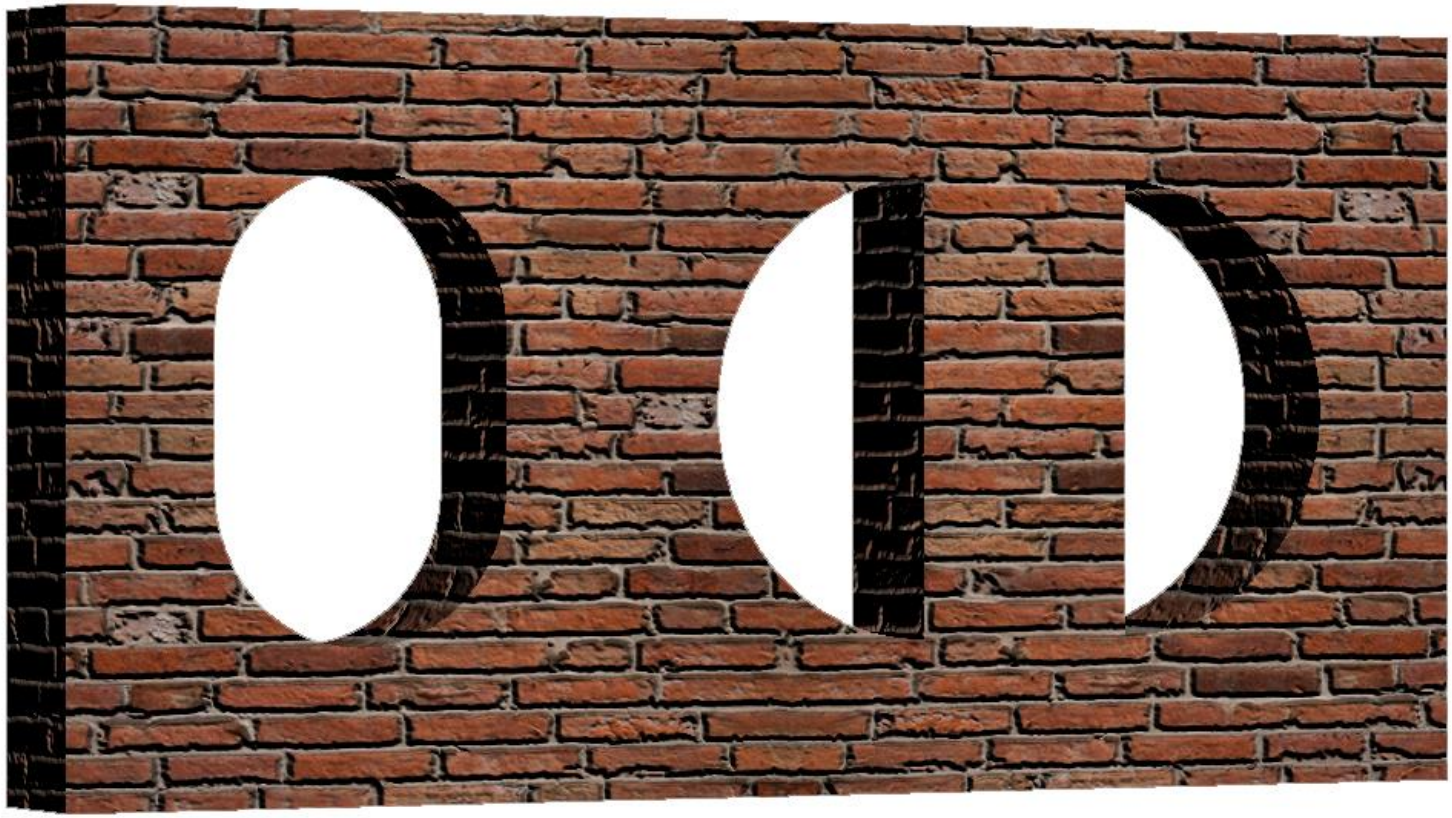
OvalArc OvalArcLeft OvalArcRight

Fensterformen in CGeoWindow



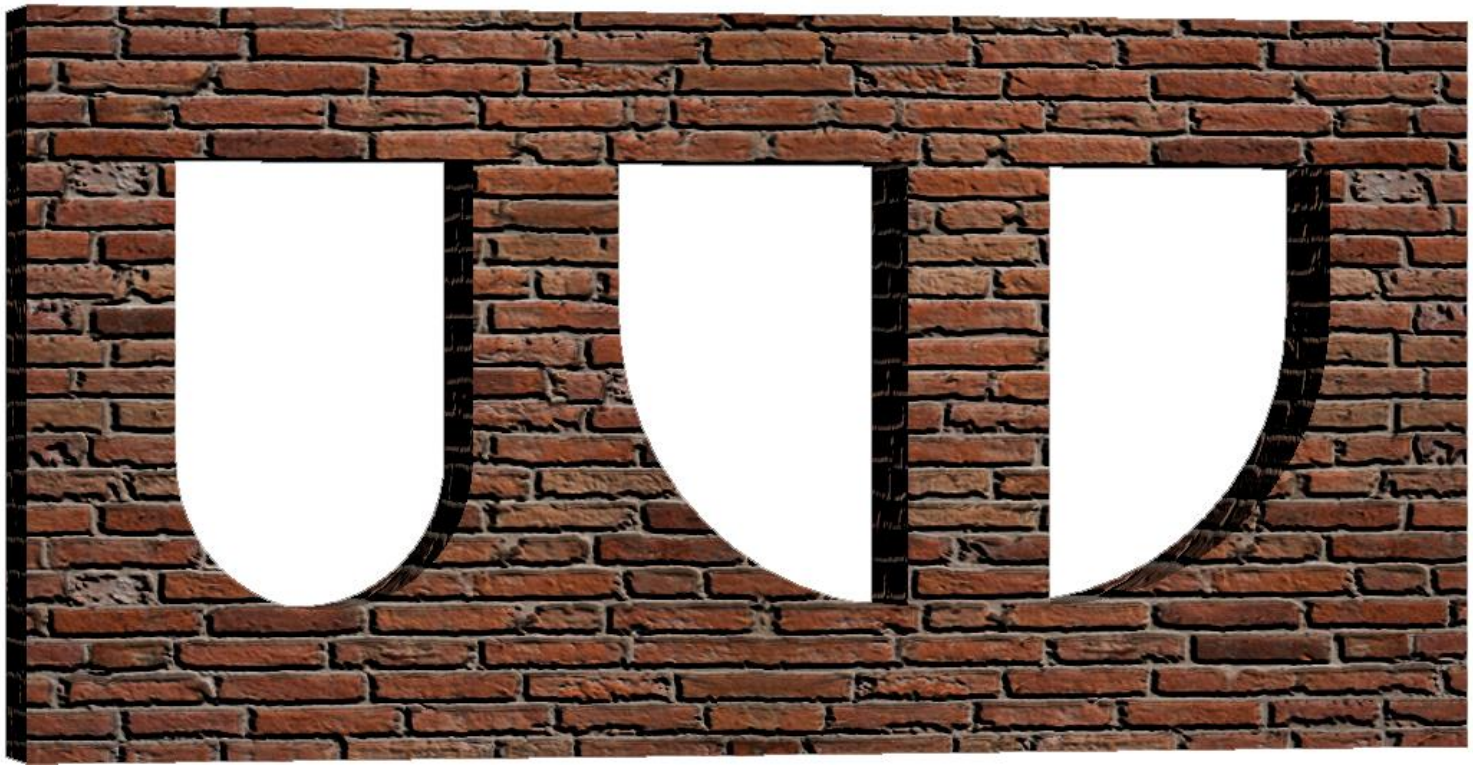
OvalU OvalULeft OvalURight

Fensterformen in CGeoWindow



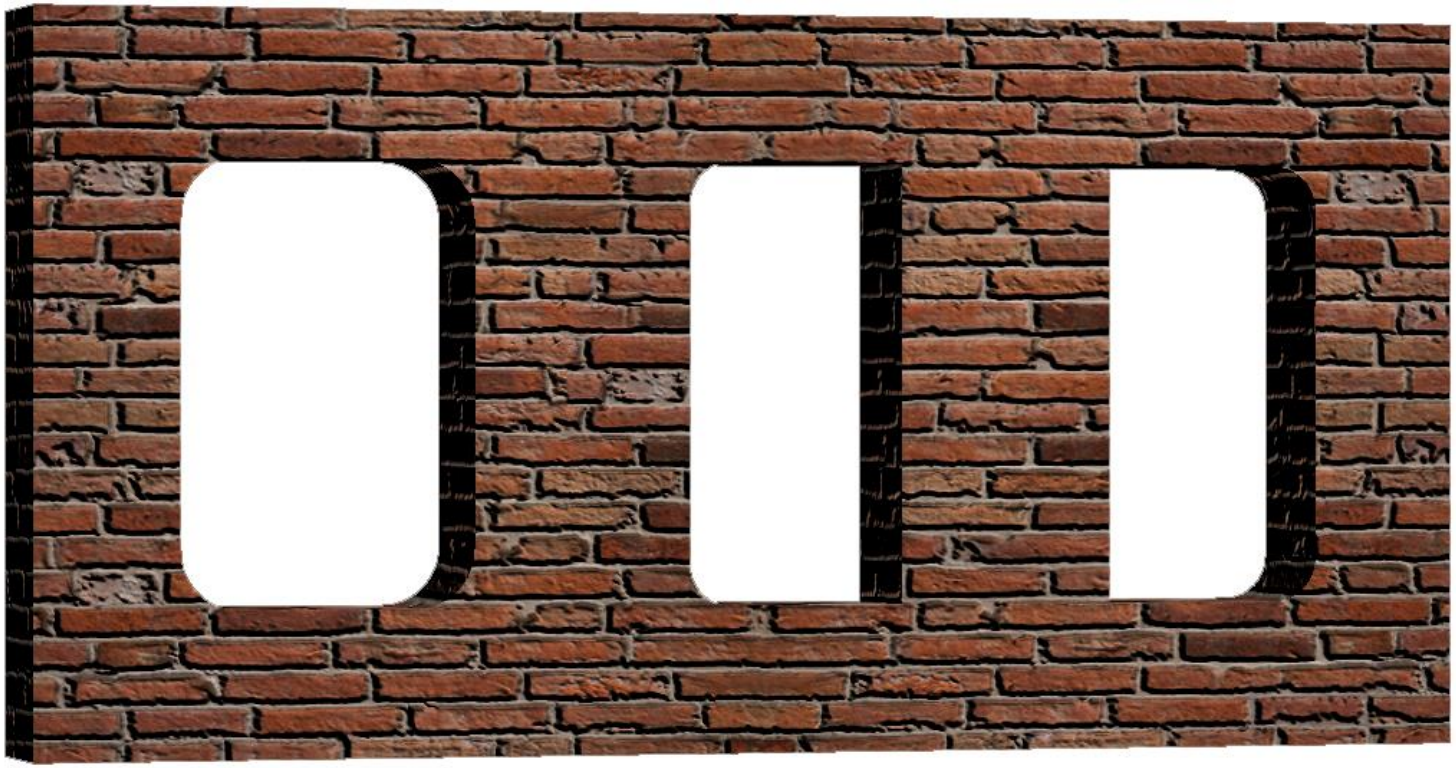
Bicircle BicircleLeft BicircleRight

Fensterformen in CGeoWindow



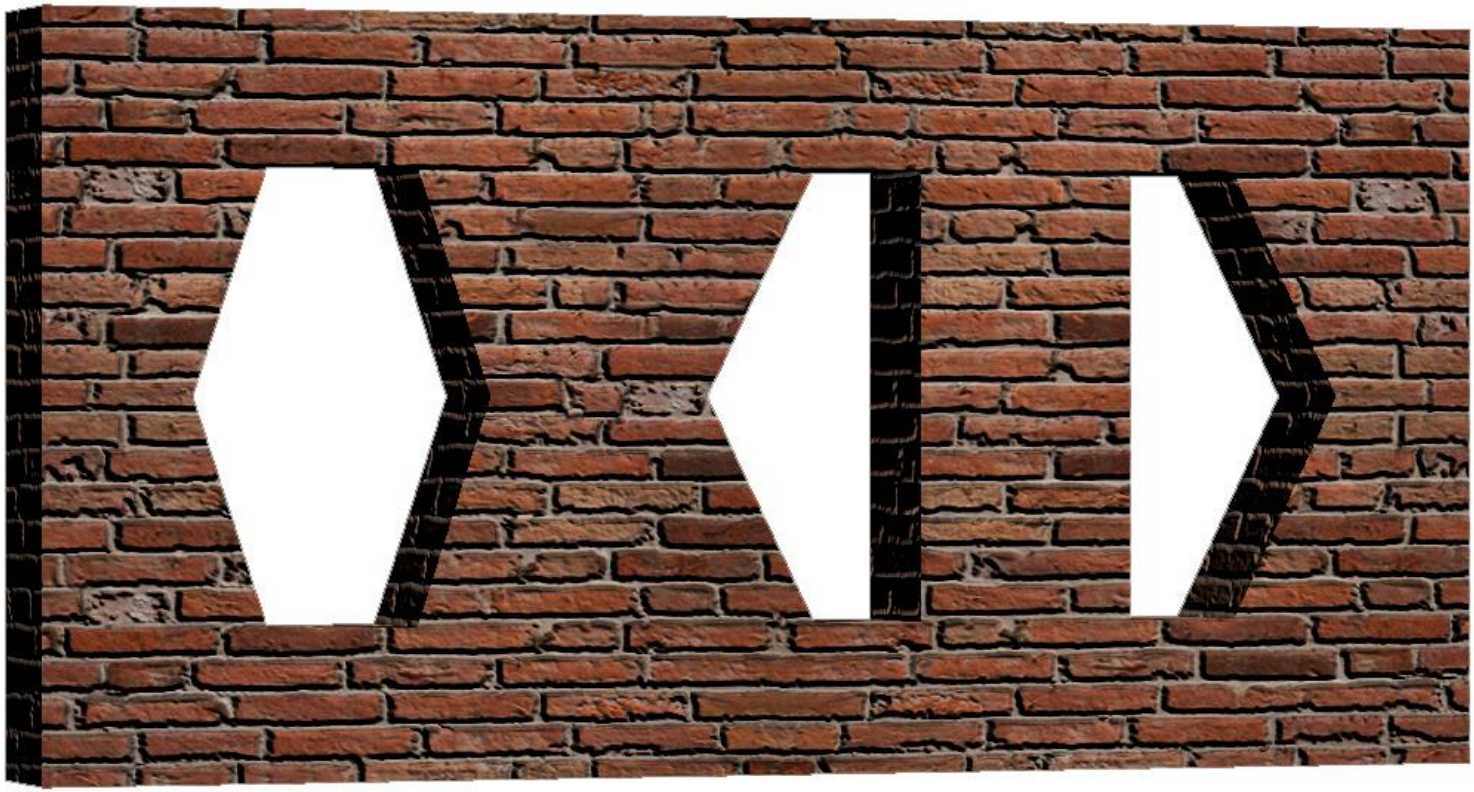
U ULeft URight

Fensterformen in CGeoWindow



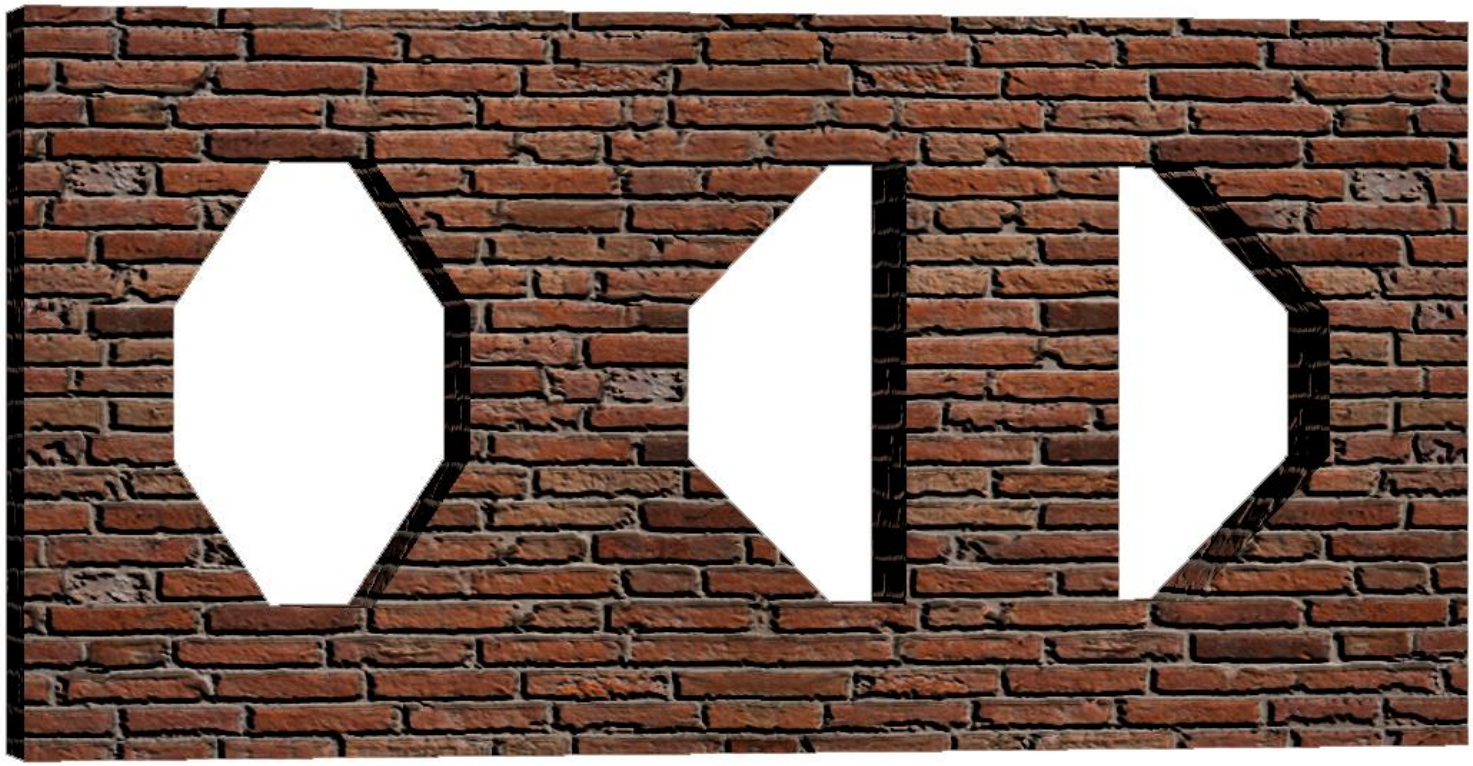
RectBeveled RectBeveledLeft RectBeveledRight

Fensterformen in CGeoWindow



Hexagon HexagonLeft HexagonRight

Fensterformen in CGeoWindow



Octagon OctagonLeft OctagonRight

Fenstermethoden in CGeoWindow

Zu jeder Fensterform existiert eine entsprechende Init-Methode in CGeoWindow, beispielsweise:

Zur Fensterform Roman existiert die Methode InitRoman und zu GothicRight existiert die Methode InitGothicRight.

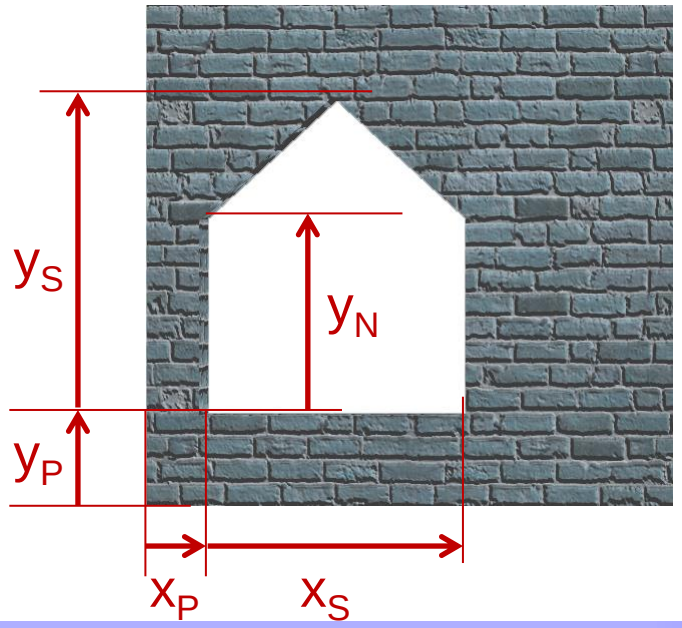
Die Fensterinitialisierungsmethoden in CGeoWindow sehen folgendermaßen aus (am Beispiel TriRect):

```
void InitTriRect(
        CFloatRect floatrect,
        bool bAbsolute = false,
        float frRectHeight = INV_PHI
                    );
```

Parameter der Init-Methoden

```
void InitTriRect(
  CFloatRect floatrect,
  bool bAbsolute = false,
  float fryHeightSection
       = INV_PHI);
```

floatrect.m_fxPos = x_P
floatrect.m_fyPos = y_P
floatrect.m_fxSize = x_S
floatrect.m_fySize = y_S
fryHeightSection = y_N



Wenn bAbsolute true ist, werden die „floatrect"-Maße in absoluten Einheiten interpretiert, ansonsten relativ zur übergeordneten Wand (Default). Der Parameter fryHeightSection wird immer relativ zu Y_S interpretiert.

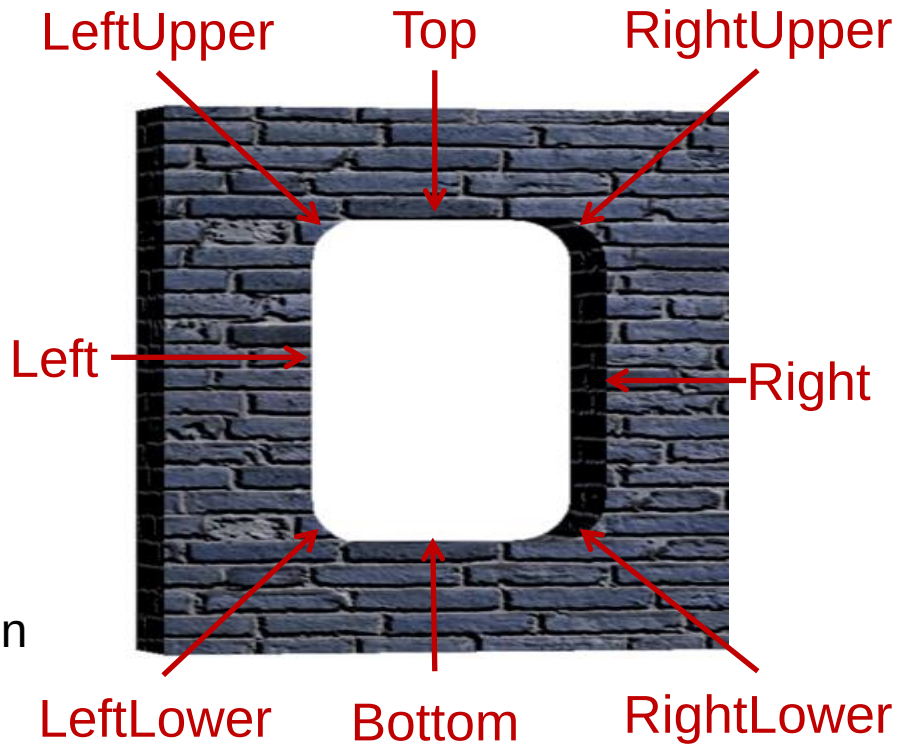
Verstecken der Fensterseiten

Mit den acht Hide-Methoden können Sie vermeiden, dass die jeweiligen Fensterseiten angezeigt werden:

```
void HideBottom();
Void HideTop();
Void HideLeft();
Void HideRight();

Void HideLeftLower();
Void HideRightLower();
Void HideLeftUpper();
Void HideRightUpper();
```

Auch diese Hide-Methoden sollten *vor* der Initalisierung aufgerufen werden!



SetInverseCurve-Methoden

Mit den SetInverseCurve-Methoden können Sie die Form der ovalen Fensterformen modifiezieren, indem Sie den entsprechenden Kurvenverlauf in der ensprechenden Fensterecke invertieren:

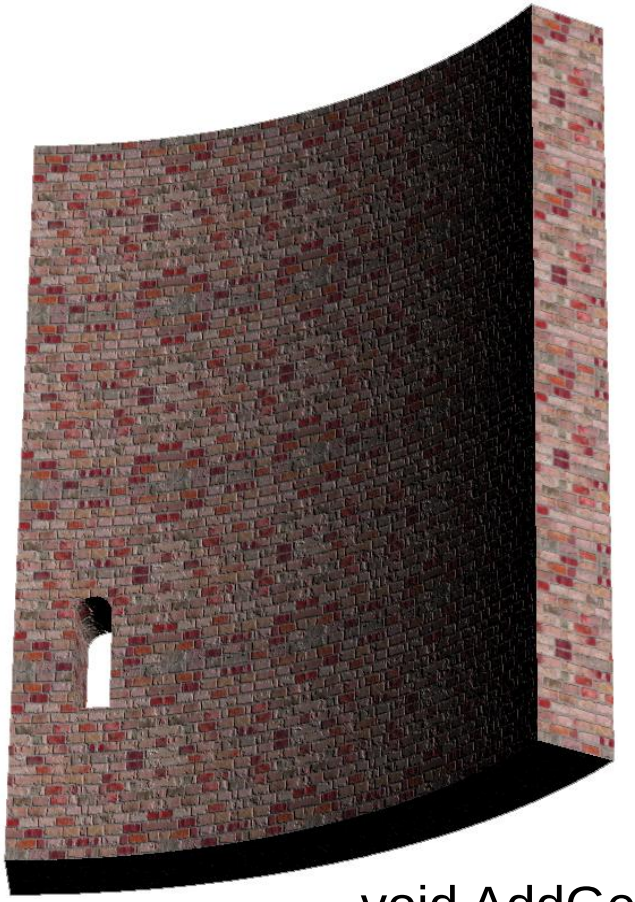
```
void SetInverseCurveLeftUpper();
void SetInverseCurveLeftLower();
void SetInverseCurveRightUpper();
void SetInverseCurveRightLower();
```

Wollen Sie alle Kurvenverläufe gleichzeitig modifizieren, steht Ihnen folgende Methode zur verfügung:

```
void SetInverseCurveAll();
```

Funktioniert nur bei Fensterformen, die irgendwo ein „Oval“ im Namen tragen, z.B. OvalArcLeft, OvalU, etc.

CGeoWall::AddGeoWindow



Mit der Verknüpfungsmethode AddGeoWindow der Klasse CGeoWall kann man ein einzelnes initialisiertes Fenster in die Wand stanzen.

```
void AddGeoWindow(CGeoWindow * pgeowindow);
```

Übung „Doppelgiebelfenster “

1

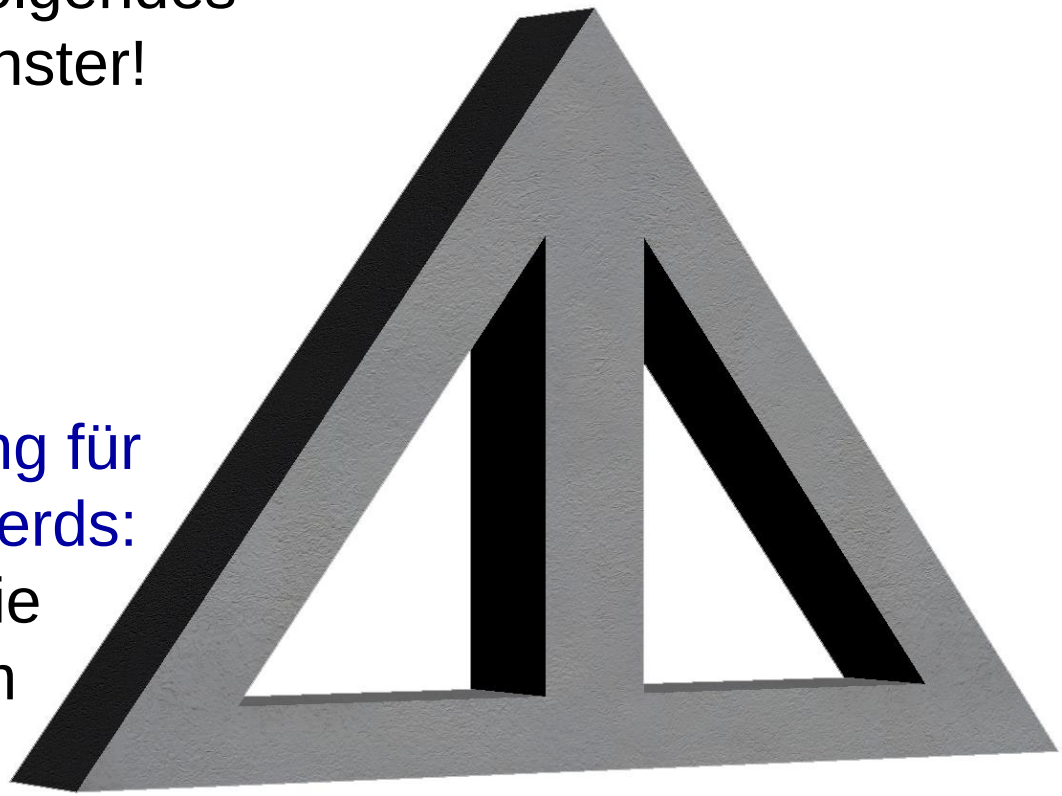
2

3

4

5

Erzeugen Sie folgendes Giebeldoppelfenster!



Freiwillige Übung für die schnellen Nerds: Versehen Sie die Wand mit einem Kniestock!

Lösung „Doppelgiebelfenster“


```
…
// Linkes Fenster erzeugen
// (false = relativ zur übergeordneten Wand):
m_zgWindowLeft.InitTriLeft(
  CFloatRect(0.15F,0.1F,0.3F,0.6F),false);
// Rechtes Fenster erzeugen  (false = relativ):
m_zgWindowRight.InitTriRight(
  CFloatRect(0.55F,0.1F,0.3F,0.6F), false);
// X-Giebelscheitelpunkt in die Wandmitte setzen:
m_zgWall.SetGable(0.5F);
// Fenster an die Wall adden:
m_zgWall.AddGeoWindow(&m_zgWindowLeft);
m_zgWall.AddGeoWindow(&m_zgWindowRight);
// Wand initialisieren:
m_zgWall.Init(1.5F,1.0F, 0.2F,&m_zm);
…
```

Übung „Bogengiebelfenster “

Erzeugen Sie folgenden gebogenen Giebel!

1

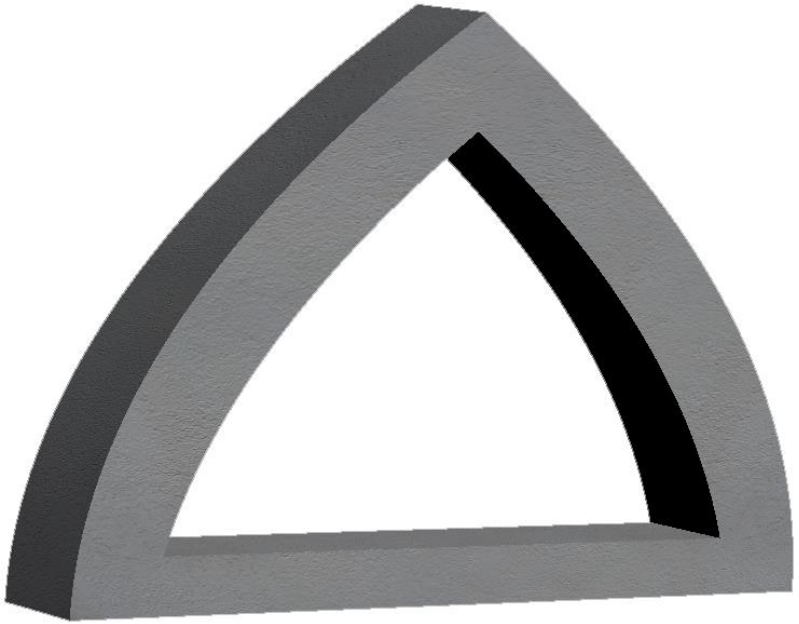
2

3

4

5

Freiwillige Übung für die schnellen Nerds:
Versehen Sie die Wand mit einem Kniestock!



Lösung „Bogengiebelfenster“


```
…
// Giebel mit Fenster erzeugen:
m_zgWindow.InitTri(
     CFloatRect(0.15F,0.1F,0.7F,0.7F),false);
m_zgWall.SetGable(0.5F);
m_zgWall.AddGeoWindow(&m_zgWindow);
m_zgWall.Init(1.5F,1.0F, 0.2F,&m_zm);
// Wand entlang der X-Achse verschieben,
// damit anschließendes Tapering symmetrisch wird
CHMat m;
m.TranslateX(-0.75F);
m_zgWall.Transform(m);
// und in Y-Richtung tapern:
m_zgWall.Subdivide(0.02);
m_zgWall.TaperY(1.0F, true, false, false);
…
```

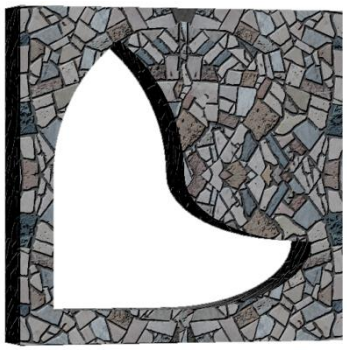
Übung „Inversfenster “

Erzeugen Sie die folgenden Fensterformen!

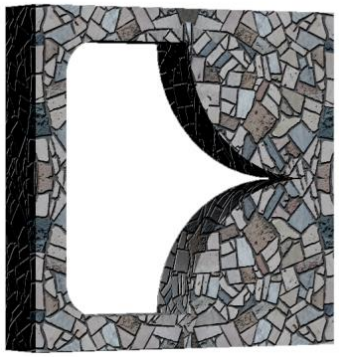
A

B

Freiwillige Übung
für die schnellen Nerds:



C



D

Lösung „Inversfenster A“


```
…
m_zgWindow.SetInverseCurveAll();
m_zgWindow.InitOvalRight(
   CFloatRect(0.1F,0.1F,0.8F,0.8F),false);
m_zgWall.AddGeoWindow(&m_zgWindow);
m_zgWall.Init(1.0F,1.0F, 0.2F,&m_zm);
…
```

Lösung „Inversfenster B“


```
…
m_zgWindow.SetInverseCurveLeftUpper();
m_zgWindow.InitOvalRight(
   CFloatRect(0.1F,0.1F,0.8F,0.8F),false);
m_zgWall.AddGeoWindow(&m_zgWindow);
m_zgWall.Init(1.0F,1.0F, 0.2F,&m_zm);
…
```

Lösung „Inversfenster C“

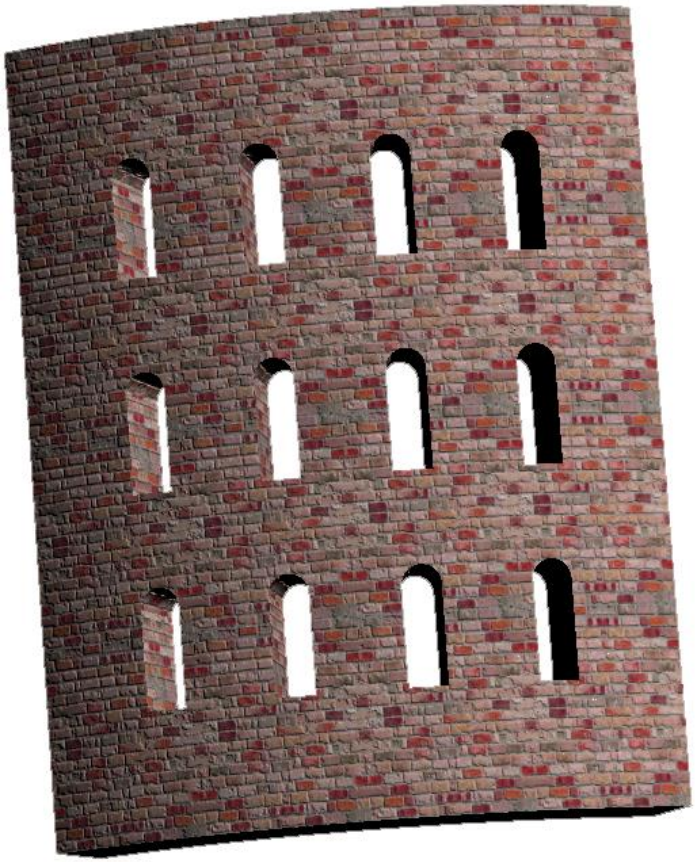

```
…
m_zgWindowLeft.InitGothic(
   CFloatRect(0.10F,0.1F,0.4F,0.8F),false,0.5F);
m_zgWindowRight.SetInverseCurveRightUpper();
m_zgWindowRight.InitOvalRight(
   CFloatRect(0.50F,0.1F,0.4F,0.4F), false);
m_zgWindowLeft.HideRight();
m_zgWindowRight.HideLeft();
m_zgWall.AddGeoWindow(&m_zgWindowLeft);
m_zgWall.AddGeoWindow(&m_zgWindowRight);
m_zgWall.Init(1.0F,1.0F, 0.2F,&m_zm);
…
```

Lösung „Inversfenster D“


```
…
m_zgWindowLeft.InitRectBeveledLeft(
    CFloatRect(0.10F,0.1F,0.4F,0.8F),false,0.5F);
m_zgWindowRight.SetInverseCurveAll();
m_zgWindowRight.InitOvalRight(
    CFloatRect(0.50F,0.1F,0.4F,0.8F), false);
m_zgWindowLeft.HideRight();
m_zgWindowRight.HideLeft();
m_zgWall.AddGeoWindow(&m_zgWindowLeft);
m_zgWall.AddGeoWindow(&m_zgWindowRight);
m_zgWall.Init(1.0F,1.0F, 0.2F,&m_zm);
…
```

1
2
3
4
5

CGeoWall::AddGeoWindows



AddGeoWindows der Klasse CGeoWall stanzt ixs*iys gleichartige Fenster rasterartig in den Mauerteil floatrect der Mauer, welches durch pgeowindow definiert wurde,

```
void AddGeoWindows(
    CGeoWindow * pgeowindow,
    CFloatRect floatrect,
    int ixs, int iys);
```

Übung „Brücke“

Erzeugen Sie folgende Brücke!

Freiwillige Übung für die schnellen Nerds:
Machen Sie die Brücke in der Mitte höher!

Lösung „Brücke“


```
…
// Romanisches Bogenfenster erzeugen:
m_zgWindow.InitRoman(
     CFloatRect(0.1F,0.2F,0.1F,0.8F),false);
// Fensterunterseite verstecken:
m_zgWindow.HideBottom();
// Sieben Fenster an den unteren Mauerrand setzen:
m_zgWall.AddGeoWindows(&m_zgWindow,
     CFloatRect(0.05F,0.0F, 0.9F, 0.8F), 7, 1);
// Die Textur in x-Richtung mehrfach wiederholen:
m_zgWall.SetTextureRepeat(7.0F,1.0F);
// Eine lange (7 Einheiten), wenig hohe (eine
// Einheit) und breite (1,2 Einheiten) Mauer
// erzeugen, darauf ein Ziegelmaterial legen :
m_zgWall.Init(7.0F,1.0F, 1.2F,&m_zm);
…
```

Nerdlösung „Brücke“


```
…
// Bogenbrücke erzeugen (siehe vorherige Lösung):
m_zgWindow.InitRoman(
     CFloatRect(0.1F,0.2F,0.1F,0.8F),false);
m_zgWindow.HideBottom();
m_zgWall.AddGeoWindows(&m_zgWindow,
     CFloatRect(0.05F,0.0F, 0.9F, 0.8F), 7, 1);
m_zgWall.SetTextureRepeat(7.0F,1.0F);
m_zgWall.Init(7.0F,1.0F, 1.2F,&m_zm);
// Polygone in X-Richtung unterteilen,
// um die  Brücke für‘s Waving vorzubereiten:
m_zgWall.SubdivideX(0.02F);
// Mit einer halben Welle x-waven,
// dabei nur die Y-Vertextupel beeinflussen:
m_zgWall.WaveX(0.3F,14.0F,0.0F,false,true,false);
…
```

Aufgabe zu Windows

Übung „Geländerbrücke“ (ohne Lösung)

1

Fügen Sie der Brücke noch ein Gittergeländer aus rostigem Metall hinzu!

2

3

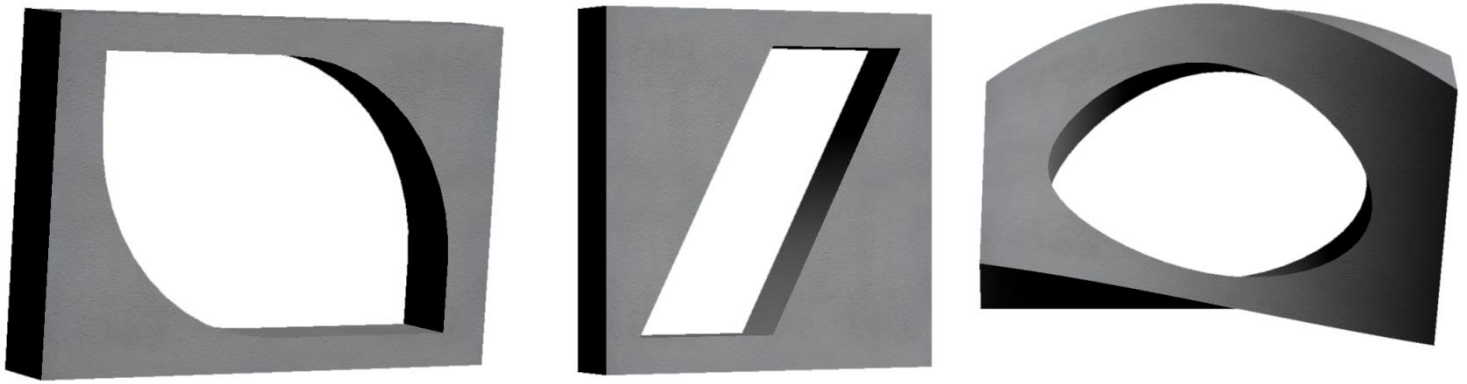
Freiwillige Übung für die schnellen Nerds:
Erzeugen Sie die berühmte „Kemptener Illerbrücke“!

4

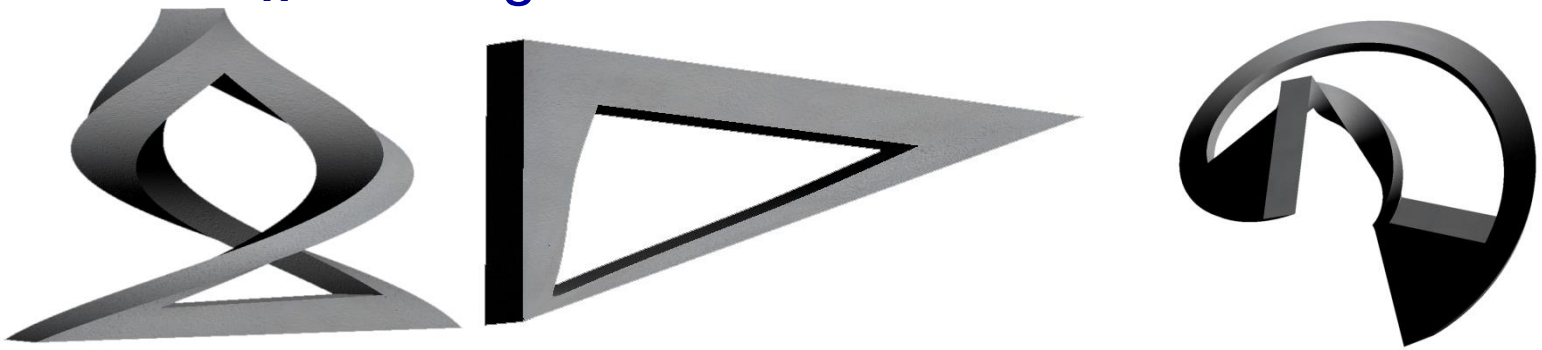
5

Übung „Spezialwände “

Erzeugen Sie die folgenden Formen!



Freiwillige Übung für die schnellen Nerds:



CGeoWindow

Add-Methoden

1

In ein Fenster kann wiederum eine Wand eingelassen werden. Diese kann wiederum Fenster enthalten. Auf diese Wiese sind komplexe Hierarchien von Fenstern und Wänden möglich, die eine Vielzahl von Geometrien zulassen.

2

Um eine Wand in ein Fenster zu setzen, existiert folgende Methode:

3

```
void AddGeoWall(CGeoWall * pgeowall);
```

4

5

Übung „Fensterrahmen“ (1/5)

1

2

3

4

5

Erzeugen Sie folgendes Holzfenster mit Beton-Fensterrahmen und Doppelverglasung!

Freiwillige Übung für die schnellen Nerds:
Fügen Sie noch Fensterläden hinzu!

Lösung „Fensterrahmen“ (2/5)


```
…
m_zgWindowLeft.InitRomanLeft(
      CFloatRect(0.10F,0.1F,0.35F,0.8F),false);
m_zgWindowRight.InitRomanRight(
      CFloatRect(0.55F,0.1F,0.35F,0.8F), false);
m_zgWallInlay.AddGeoWindow(&m_zgWindowLeft);
m_zgWallInlay.AddGeoWindow(&m_zgWindowRight);
m_zgWallInlay.SetTextureRepeat(3.0F,3.0F);
m_zgWallInlay.Init(1.0F,1.0F, 0.2F,&m_zmWood);
m_zgWallGlass.Init(1.0F,1.0F,0.1F,&m_zmGlass);
m_zgWindow.AddGeoWall(&m_zgWallInlay);
m_zgWindow.AddGeoWall(&m_zgWallGlass);
m_zgWindow.InitRoman(
      CFloatRect(0.1F,0.1F,0.8F,0.8F));
…
```

Lösung „Fensterrahmen“ (3/5)

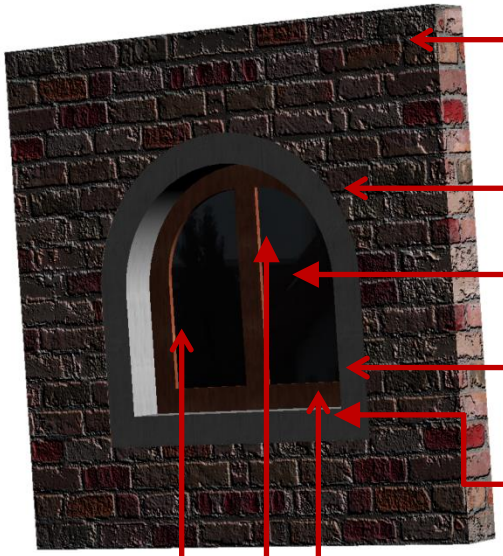

```
…
m_zgWallFrame.AddGeoWindow(&m_zgWindow);
m_zgWallFrame.SetTextureRepeat(3.0F,3.0F);
m_zgWallFrame.Init(1.0F,1.0F,0.99F,&m_zmFrame);
m_zgWindowFrame.AddGeoWall(&m_zgWallFrame);
m_zgWindowFrame.InitRoman(CFloatRect(0.2F,0.2F,0.6F,
0.6F));
m_zgWall.AddGeoWindow(&m_zgWindowFrame);
m_zgWall.Init(2.0F,2.0F, 0.2F,&m_zmBricks);
…
```

Lösung „Fensterrahmen“ (4/5)

Texturtabelle	Umrandung	Wand	Rahmen
Image textures			
Bump textures			
Specular textures			

Lösung „Fensterrahmen“ (5/5)

1
2
3
4
5



CGeoWall
m_zgWall;

CGeoWindow
m_zgWindowFrame;

CGeoWall
m_zgFrame;

CGeoWindow
m_zgWindowInlay;

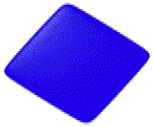
CGeoWall
m_zgWallInlay;

CGeoWall
m_zgWallGlass;

CGeoWindow
m_zgWindowLeft;

CGeoWindow
m_zgWindowRight;

Kapitel 4



Was ist ein Wing?

1. Ein Wing wird durch die Klassse CGeoWing repräsentiert. Es stellt ein Gebäudeflügel dar. Ein Gebäudeflügel besteht aus mehreren Arten von Walls.

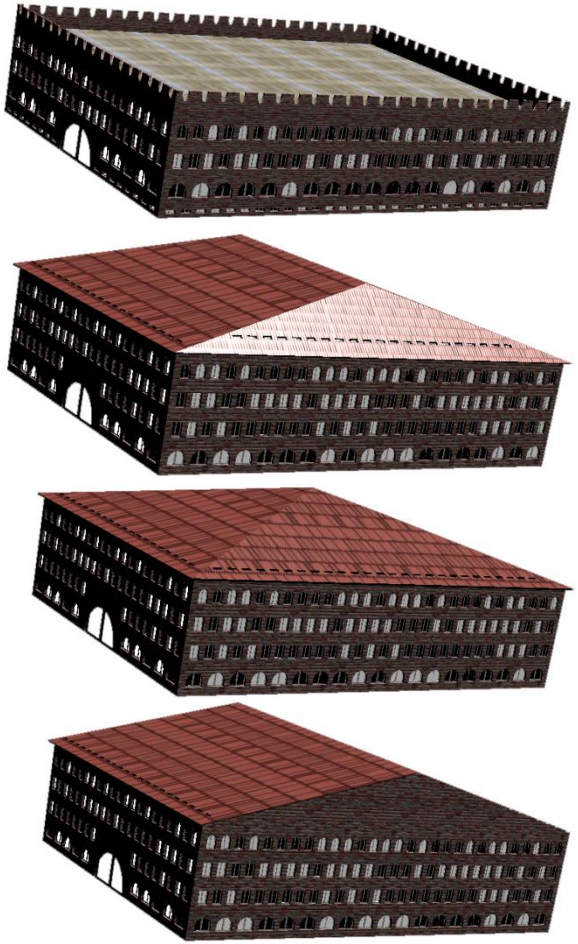
2. Ein Wing kann Unterwings enthalten.

3. Alle Eigenschaften des Vaterwings vererben sich automatisch auf den Sohn, die Eigenschaften können jedoch vom Sohn überschrieben werden (Polymorphie).

4. Dies erlaubt eine objektorientierte und schnelle Erstellung großer Architekturkomplexe.

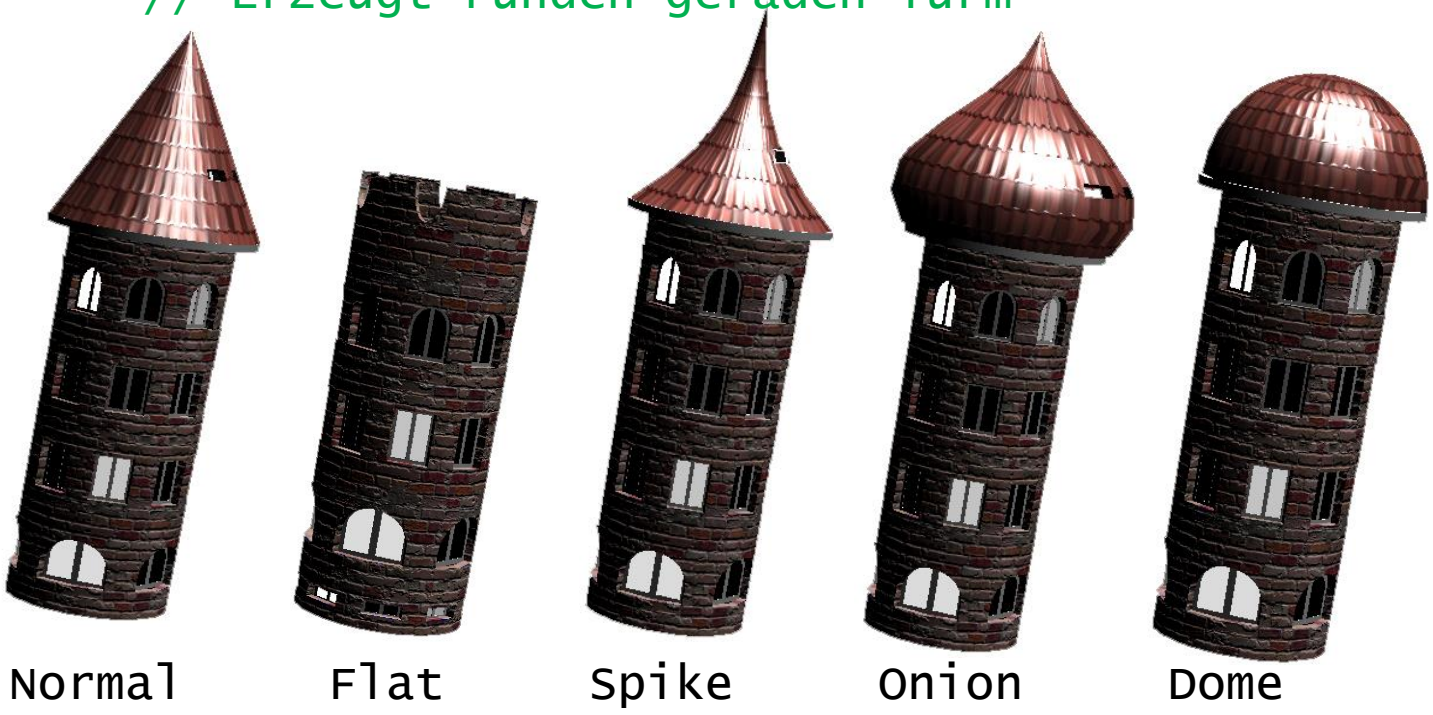
CGeoWing-Initialisierungsmethoden

```
void InitRectFlat();
        // erzeugt rechteckiges
        // Architekturgebilde
        // mit Flachdach
void InitRectMans();
        // erzeugt rechteckiges
        // Architekturgebilde
        // mit Mansardendach
void InitRectTent();
        // erzeugt rechteckiges
        // Architekturgebilde
        // mit Zeltdach
void InitRectSaddle();
        // erzeugt rechteckiges
        // Architekturgebilde
        // mit Satteldach
```



CGeoWing-Initialisierungsmethoden

```
void InitRoundTower
        (EGeoWingTowerRoof eKind =
        eGeoWingTowerRoof_Normal);
        // Erzeugt runden geraden Turm
```



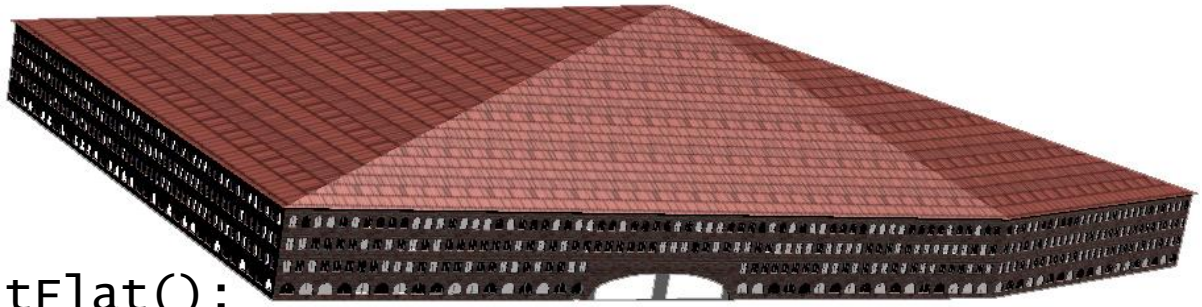
CGeoWing-Initialisierungsmethoden

```
void InitPolyFlat();
        // erzeugt Gebäude bzw. Gebäudeflügel
        // mit n gleichseitgen Kanten
        // und Flachdach
```

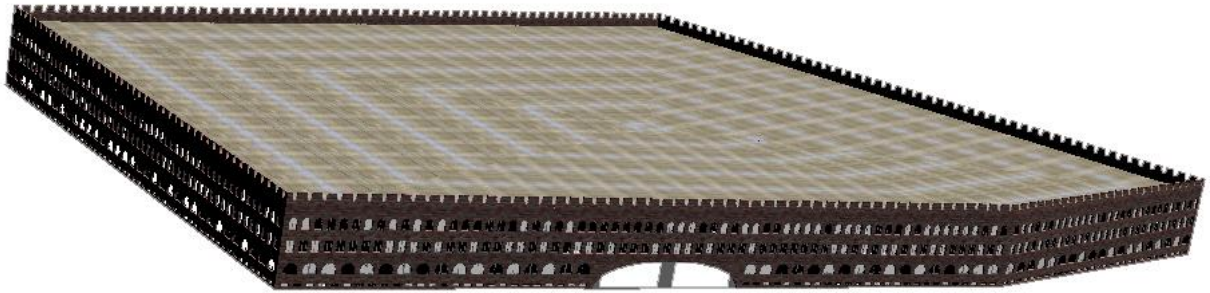
```
void InitPolyTent();
        // erzeugt Gebäude bzw. Gebäudeflügel
        // mit n gleichseitgen Kanten
        // und Zeltdach
```

CGeoWing-Initialisierungsmethoden

```
void InitPrintTent();
        // erzeugt Gebäude bzw. Gebäudeflügel
        // mit beliebigen Grundriss und Zeltdach
```

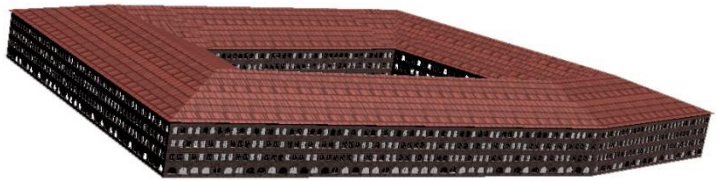


```
void InitPrintFlat();
        // erzeugt Gebäude bzw. Gebäudeflügel
        // mit beliebigen Grundriss und Flachdach
```

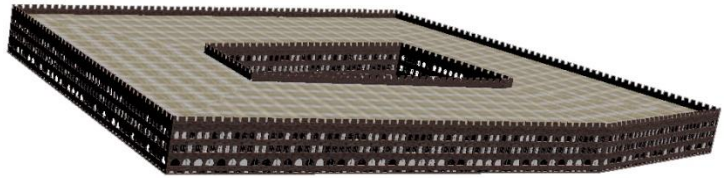


CGeoWing-Initialisierungsmethoden

1
```
void InitYardRoof();
// erzeugt Architekturgebilde
// mit beliebigen Grundriss,
// Satteldach und Innenhof
```



2
3
```
void InitYardFlat();
// erzeugt Architekturgebilde
// mit beliebigen Grundriss,
// Flachdach und Innenhof
```



4
5
```
void InitYardRutsh();
// erzeugt Architekturgebilde
// mit beliebigen Grundriss,
// Schrägdach und Innenhof
```

CGeoWing-Grundrissmethoden

```
void SetSizeRect(
        float fWidth,
        float fDepth,
        float fHeightFacade);
        // Setzt die Ausmaße eines rechteckigen
        // Standardgebäudeflügels
void SetSizeTower(
        float fRadius,
        float fHeightFacade);
        // Setzt die Ausmaße eines runden Turmes
void SetSizePrint(
        float fDepth,
        float fHeightFacade);
        // setzt die Ausmaße für ein
        // beliebiges Grundrissgebäude
```

CGeoWing-Grundrissmethoden

```
void SetHeightBalustrade(
        float fHeightBalustrade);
        // Setzt die Geländerhöhe für
        // Flachdächer und Balkone
```

CGeoWing-Parametrisierungsmethoden

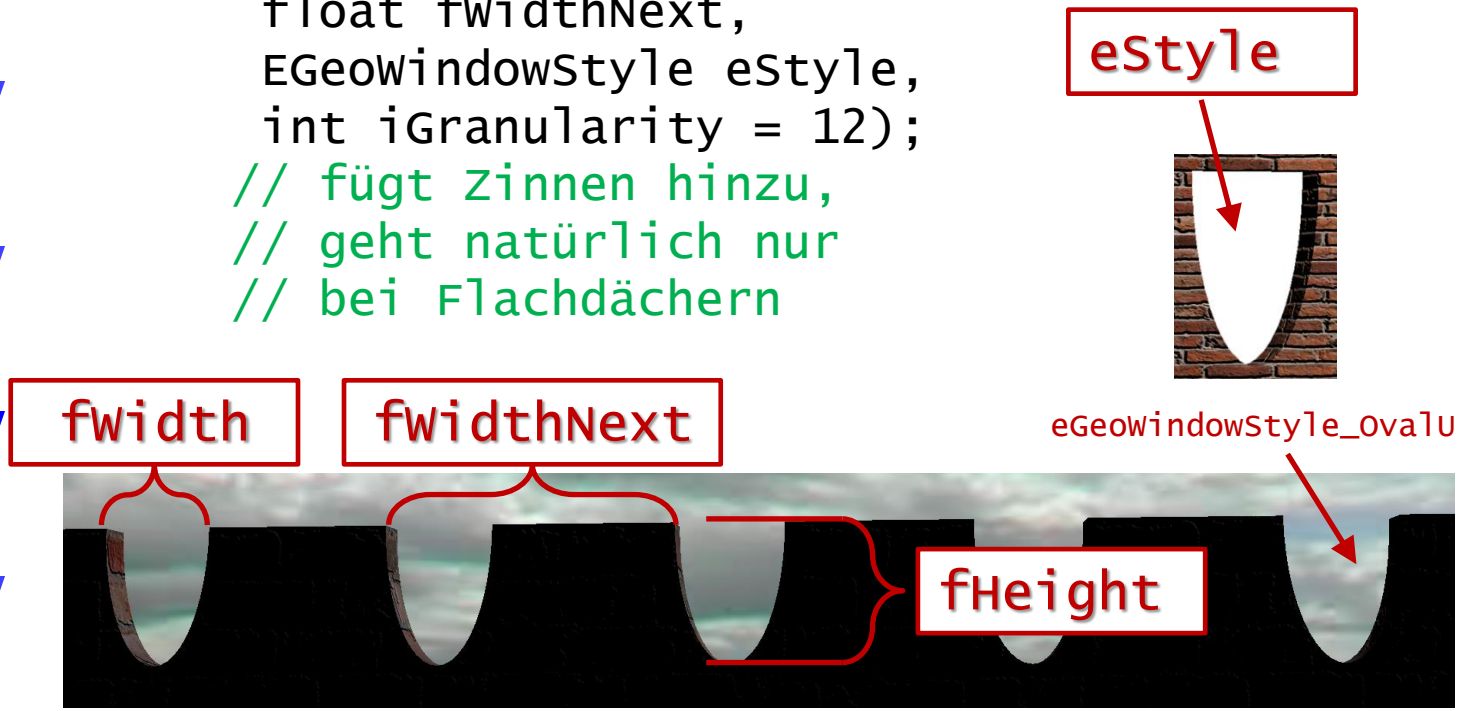
```
void SetOverroof
       (float fOverroofDepth,
       float fOverroofHeight);
// setzt den Dach-Überhang fOvererroofDepth
// und die Dachversatztiefe fOverroofHeight
```

CGeoWing-Parametrisierungsmethoden

```
void SetInner(bool bInner);
// true, wenn Innenwände angezeigt werden sollen,
// ansonsten false
```

CGeoWing-Zinnenmethoden

```
void SetBattlement
        (float fHeight,
         float fWidth,
         float fWidthNext,
         EGeoWindowStyle eStyle,
         int iGranularity = 12);
       // fügt Zinnen hinzu,
       // geht natürlich nur
       // bei Flachdächern
```



CGeoWing-Grundrissmethoden

```
void SetPrint(CHVectors * phvectorsPrint);
      // Legt Grundriss des Gebäudes fest
      // funktioniert natürlich nur auf
      // Initialisierungsmethoden, die mit
      // Grundrissen arbeiten, also die
      // InitYard- Und InitPrint-Methoden
```

CGeoWing-Fenstermethoden

1

```
void AddGeoWindow(CGeoWindow * pgeowindow,
int iWall, EGeoWingPart ePart = eGeoWingPart_Facade);
// Stanzt ein einzelnes Fenster an die Wand mit der Nummer iWall
```

2

3

```
void AddGeoWindows(CGeoWindow * pgeowindow, int iWall,
CFloatRect floatrect, int ixs, int iys, EGeoWingPart ePart =
eGeoWingPart_Facade);
// Stanzt ixs mal iys Fenster in den Bereich der Wand mit der
Nummer iWall, der mit floatrect angegeben ist
```

4

5

CGeoWing-Fenstermethoden

1. void AddGeoWindowsRandom(CGeoWindow * pgeowindow, CGeoWindow * pgeowindowDoor, int iWallDoor);
   // Versucht, die Fenster automatisch zu platzieren
2. void AddRandomDoor(CGeoWindow * pgeowindow, int iWallDoor);
   // Fügt eine Tür hinzu, die automatisch platziert wird
3. void AddRandomWindow(CGeoWindow * pgeowindow);
   // Fügt ein Fenster hinzu, das automatisch platziert wird
4.
5.

CGeoWing-Fenstermethoden (1/2)

1
```
void AddRandomWindowHighest(CGeoWindow * pgeowindow);
// Fügt ein Fenster hinzu, das automatisch in die oberste
// Fensterzeile platziert wird
```

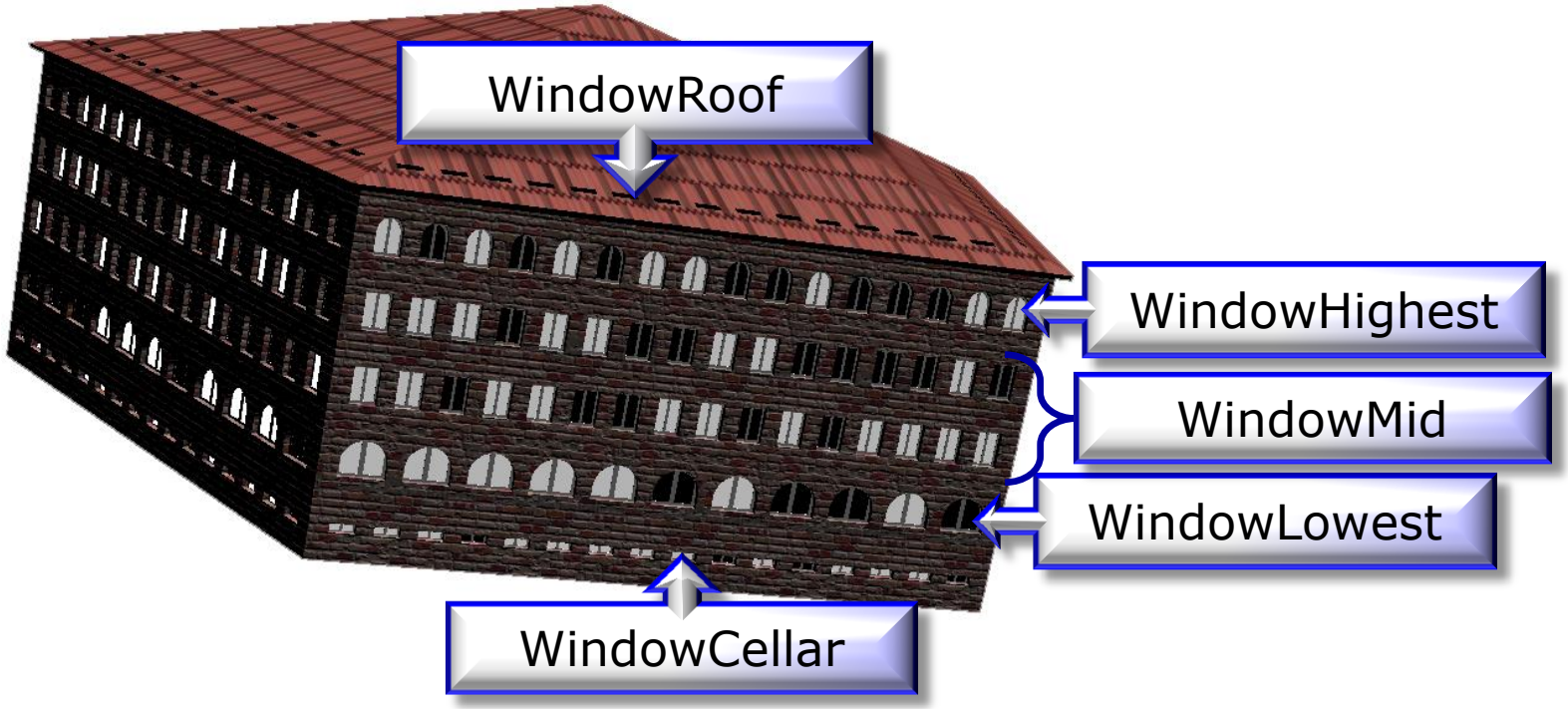
2
```
void AddRandomWindowLowest(CGeoWindow * pgeowindow);
// Fügt ein Fenster hinzu, das automatisch in die unterste
// Fensterzeile platziert wird
```

3
```
void AddRandomWindowMid(CGeoWindow * pgeowindow);
// Fügt ein Fenstertyp hinzu, das automatisch in die mittleren
// Fensterzeilen platziert wird
```

4
```
void AddRandomWindowRoof(CGeoWindow * pgeowindow);
// Fügt ein Fenstertyp hinzu, das automatisch auf die Dächer
// platziert wird
```

5
```
void AddRandomWindowCellar(CGeoWindow * pgeowindow);
// Fügt ein Fenstertyp hinzu, die automatisch als Kellerfenster
// platziert wird
```

CGeoWing-Fenstermethoden (2/2)



Übung „Gebäude“

Erzeugen Sie ein möglichst komplexes Gebäude Ihrer Wahl!

Freiwillige Übung für die schnellen Nerds:

Lassen Sie einige Fenster in der Nacht leuchten, damit das Gebäude bewohnt aussieht!

Kapitel 5

Anwendung weiterer Methoden

1. Um weitere Gebäudestrukturen zu erzeugen, können Sie die erzeugten Architekturteile mit Barr-Modellierung, Waving, Rippleing, Magnets, etc. verfeinern.

2. Dieser Vorgang wird als Postmodellierung bezeichnet.

3. Im Folgenden einige Beispiele dafür:

4.

5.

Postmodellierung mit Y-Rippleing

```
…
m_zgWingMain.SetThickness(0.2);
m_zgWingMain.SetHeightRoof(5.0F);
m_zgWingMain.SetOverroof(0.5F,0.1F);
m_zgWingMain.SetSizeTower(2,20.1);
…
m_zgWingMain.InitPolyTent();
m_zgWingMain.Subdivide(1.0);
m_zgWingMain.RippleY(0.3,5,0);
…
```

Postmodellierung mit Y-Rippleing

```
…
m_zgWingMain.SetThickness(0.2);
m_zgWingMain.SetHeightRoof(5.0F);
m_zgWingMain.SetOverroof(0.5F,0.1F);
m_zgWingMain.SetSizeTower(2,20.1);
…
m_zgWingMain.InitPolyTent();
m_zgWingMain.Subdivide(0.5F);
m_zgWingMain.RippleY(-0.1F,5.0F,0.0F);
…
```

GAME OVER