




# Computer-Graphik 1

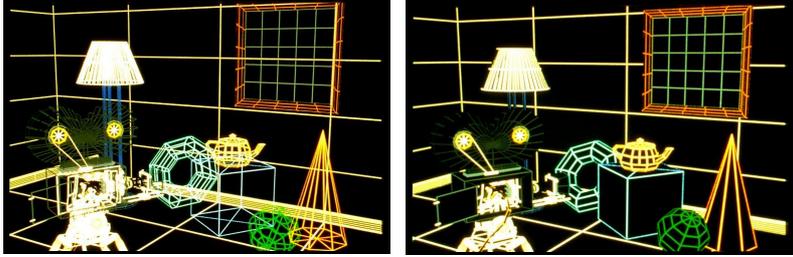
## Visibility Computations I - Hidden Surface Determination

G. Zachmann  
Clausthal University, Germany  
[zach@in.tu-clausthal.de](mailto:zach@in.tu-clausthal.de)



## Motivation

- **Verdeckung** entsteht, wenn mehrere Objekte bei der Abbildung von 3D nach 2D (teilweise) die gleichen Bildschirmkoordinaten aufweisen (*Projektionsäquivalenz*)
- **Sichtbar** ist das dem Auge am nächsten liegende Objekt
- Ist dieses Objekt durchsichtig (transparent), wird der dahinter liegende Punkt auch sichtbar, usw.



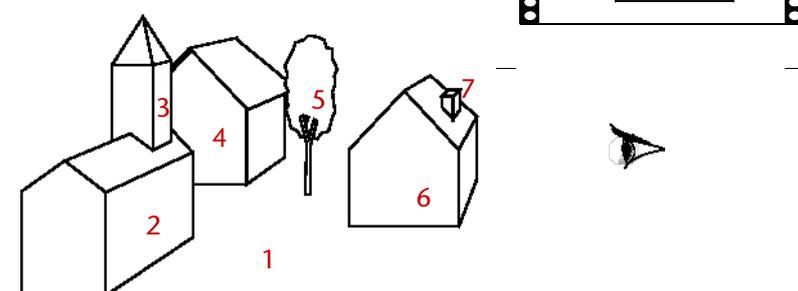
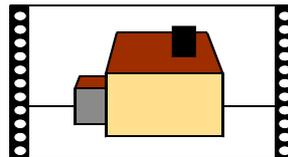
G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 2

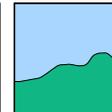
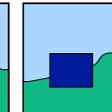
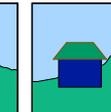
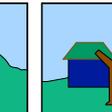
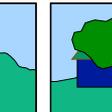
- Es gibt 2 große Problemklassen innerhalb des Bereichs "Visibility Computations"
  1. **Verdeckungsrechnung:** welche Polygone (oder Teile) werden von anderen verdeckt?
    1. Bezeichnungen: *Hidden Surface Elimination* (früher auch *Hidden Line Elimination*), *Visible Surface Determination*
    2. **Culling:** welche Polygone / Objekte können gar nicht sichtbar sein? (z.B., weil sie sich hinter dem Viewpoint befinden)
- Achtung: die Grenzen sind fließend
  - Tendentieller Unterschied: bei HSE geht es eher darum, überhaupt ein **korrektes Bild** zu rendern, bei Culling geht es eher um eine **Beschleunigung** des Renderings großer Szenen

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 3

## Die einfachste Idee: Der Painter's Algorithm

- Idee: Zeichne das Bild wie ein Maler
  - Zuerst den Hintergrund
  - Dann Objekte von hinten nach vorne

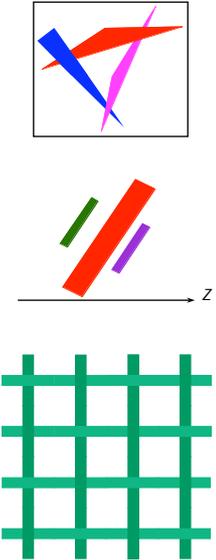



G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 5

## Probleme

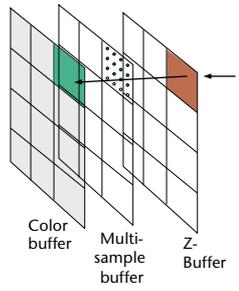
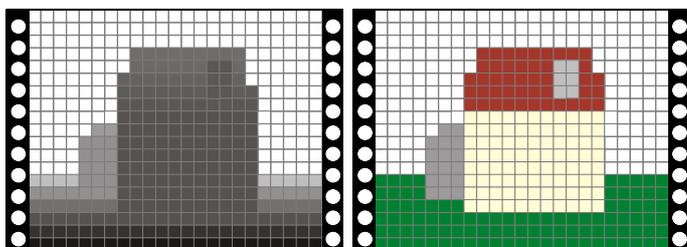
- Es gibt Fälle, in denen eine korrekte Sortierung nicht existiert!
  - Oder nicht klar ist ...
- Eine Lösung wäre evtl. eine Zerlegung der Polygone — aber ...
- Diese Zerlegung ist (im Prinzip) abhängig vom Viewpoint; und ...
- Bei einer Szene mit  $n$  Polygonen können  $O(n^2)$  sichtbare Fragmente entstehen



G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 6

## Die Standard-Lösung heute: der Z-Buffer

- Zusätzlich zum Color Buffer (R, G, B)
- Speichert pro Pixel den Abstand  $z$  zur Kamera
- Pixel wird geschrieben, wenn  $Z$  kleiner ist als der Wert im Z-Buffer

Z-Buffer Color Buffer

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 7

### Beispiel

(a)

R	R	R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R	R	R

+

5	5	5	5	5	5	5	5		
5	5	5	5	5	5	5			
5	5	5	5	5					
5	5	5	5						
5	5	5							
5	5								
5									
5									

=

5	5	5	5	5	5	5	5	R							
5	5	5	5	5	5	5		R	R						
5	5	5	5	5	5			R	R	R					
5	5	5	5					R	R	R	R				
5	5	5						R	R	R	R	R			
5	5							R	R	R	R	R	R		
5								R	R	R	R	R	R	R	
5								R	R	R	R	R	R	R	R

(b)

5	5	5	5	5	5	5	R							
5	5	5	5	5	5		R	R						
5	5	5	5	5			R	R	R					
5	5	5					R	R	R	R				
5	5						R	R	R	R	R			
5							R	R	R	R	R	R		
5							R	R	R	R	R	R	R	
5							R	R	R	R	R	R	R	R

+

8														
7	8													
6	7	8												
5	6	7	8											
4	5	6	7	8										
3	4	5	6	7	8									

=

5	5	5	5	5	5	5	5	R							
5	5	5	5	5	5	5		R	R						
5	5	5	5	5				R	R	R					
5	5	5	8					R	R	R	R				
5	5	8						R	R	R	R	R			
4	5	6	7	8				R	R	R	R	R	R		
3	4	5	6	7	8			R	R	R	R	R	R	R	
								R	R	R	R	R	R	R	R

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 8

### Z-Buffer Pseudo-Code

```

for all pixels in window:
    framebuffer[x,y] = BACKGROUND_COLOR; zbuffer[x,y] = ∞;
for every triangle:
    compute projection & color at vertices
    setup edge equations
    compute bbox, then clip bbox to screen limits
    for all pixels x,y in bbox:
        increment edge equations
        compute Z of current pixel / point
        compute current color c (incrementally)
        if all edge equations > 0: // pixel is in triangle
            if current Z < zBuffer[x,y]: // pixel is visible
                framebuffer[x,y]= c
                zBuffer[x,y] = current Z
    
```

▪ Funktioniert auch in schwierigen Fällen:

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 9

## Berechnung des Z-Wertes bei der Scan-Conversion

$$z_a = z_1 + \frac{y_s - y_1}{y_2 - y_1}(z_2 - z_1) \quad z_b = z_1 + \frac{y_s - y_1}{y_3 - y_1}(z_3 - z_1)$$

$$z_p = z_a + \frac{x_p - x_a}{x_b - x_a}(z_b - z_a)$$

Oder:  $z_p = \alpha z_1 + \beta z_2 + \gamma z_3$   
 wobei  $\alpha, \beta, \gamma$  wie gehabt inkrementell im Algorithmus von Pineda berechnet werden

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 10

## Der Z-Buffer in OpenGL

- Fenster mit Z-Buffer anmelden (hier am Bsp. von GLUT)
 

```
glutInitDisplayMode( GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH );
```
- Einschalten
 

```
glEnable( GL_DEPTH_TEST );
```
- Wichtig: nicht nur Bildspeicher, sondern auch Z-Buffer löschen!
 

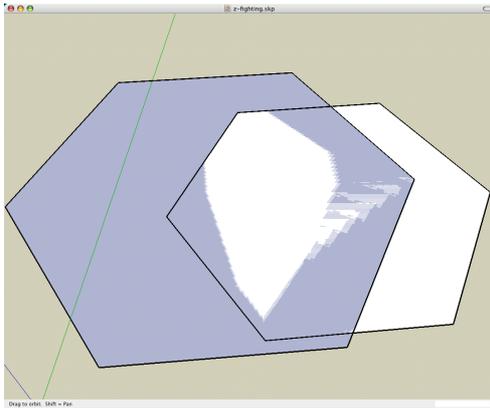
```
glClear( GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT );
```

- Achtung: unter Qt ist (1) und (2) per Default angeschaltet
  - Mehr Info unter <http://doc.trolltech.com/4.2/qglformat.html>
  - Beispiel zu QtFormat im "OpenGL/Qt-Programmbeispiel" auf der Homepage der Vorlesung

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 11

## Z-Fighting

- Wegen der begrenzten Auflösung des Z-Buffers kommt es bei koplanaren oder fast koplanaren Polygonen zum sog. **Z-Fighting**:



G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 13

## Bewertung

- Komplexität des Algorithmus':  $O(n)$ , mit  $n$  = Anzahl Polygone
  - Kein zusätzlicher Aufwand, z.B. durch Sortieren (z.B.  $O(n \log n)$ )
- Eigentlich:  $O(n+p)$ , wobei  $p$  = # geschriebene Pixel (kann unter Umständen viel größer als Anzahl sichtbarer Pixel sein!)
- Lässt sich ideal in Hardware implementieren:
  - Parallelisierung ohne Kommunikations-Overhead
  - Keine komplizierte "Logik" (wenige "if"s)
  - Keine komplizierten Datenstrukturen zu traversieren (z.B. verzeigte Strukturen, z.B. Bäume)
- Nachteile:
  - Pro Pixel kann nur ein Primitiv gespeichert werden
    - Einige fortgeschrittene Effekte, z.B. Transparenz, benötigen aber alle Primitive
  - Genauigkeit des Z-Buffers ist oft stark beschränkt (image space vs. object space)
    - Auch heute noch manchmal 16-Bit Integer-Werte, um Speicherplatz zu sparen

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 14

## Hierarchischer Z-Buffer (HZB) [Greene, 1993]

- Idee: „Z-Pyramide“
  - einfacher Z-Buffer = höchste Auflösung
  - weitere Levels durch Zusammenfassen von jeweils 4 Pixel
  - z-Wert auf max. z-Wert setzen

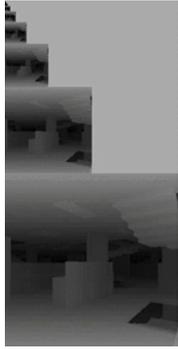
7	1	0	6
0	3	1	2
3	9	1	2
9	1	2	2

farthest value

7	6
9	2

farthest value

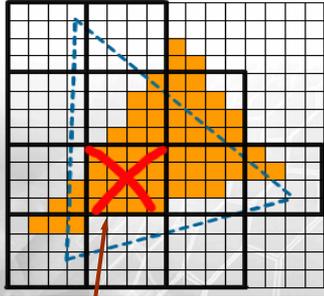
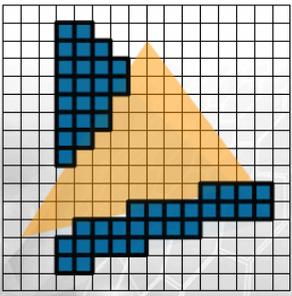
9
---

G. Zachmann Computer-Graphik 1 - WS 09/10
Hidden Surface Determination / Buffers 15

## Beispiel

- Orangenes Dreieck bereits gezeichnet
- Blaues Dreieck soll dahinter gezeichnet werden

vollständig verdeckt →  
verwerfe gesamten Block

G. Zachmann Computer-Graphik 1 - WS 09/10
Hidden Surface Determination / Buffers 16

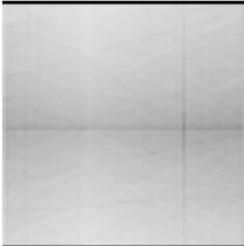
## Vergleich

- Definition **Depth-Complexity**:
  - Eigentlich: Anzahl Polygone, die "hinter" einem Pixel liegen
  - Hier: Anzahl z-Tests pro Pixel

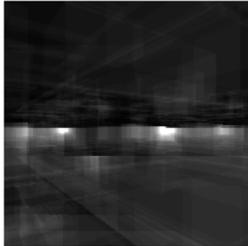
540 Mio Polygone



Depth complexity mit  
einfachem Z-Buffer



Depth complexity mit HZB



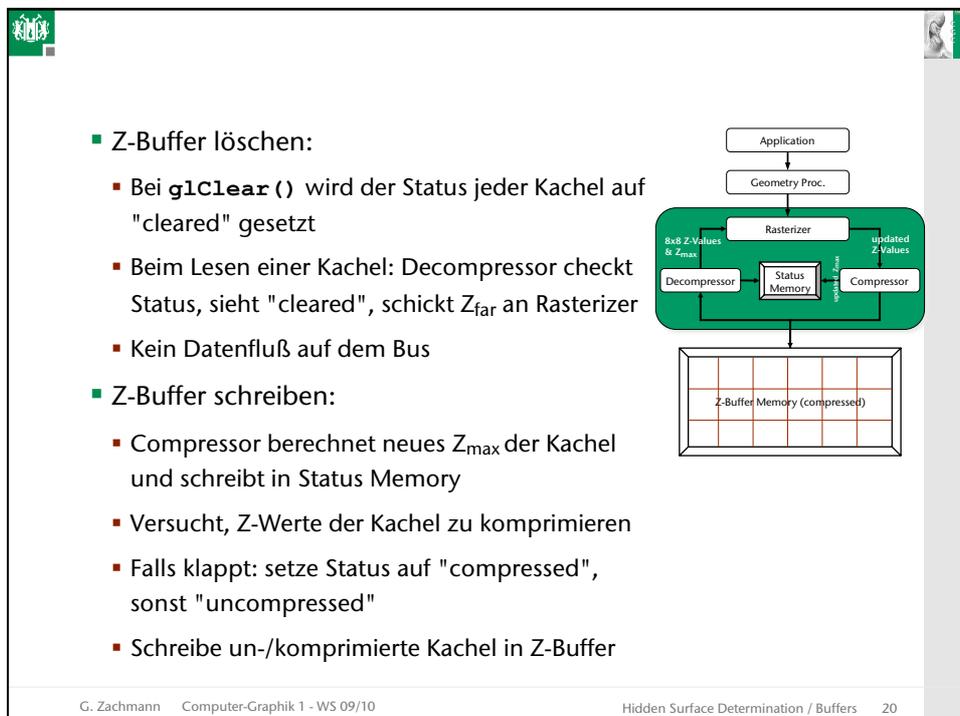
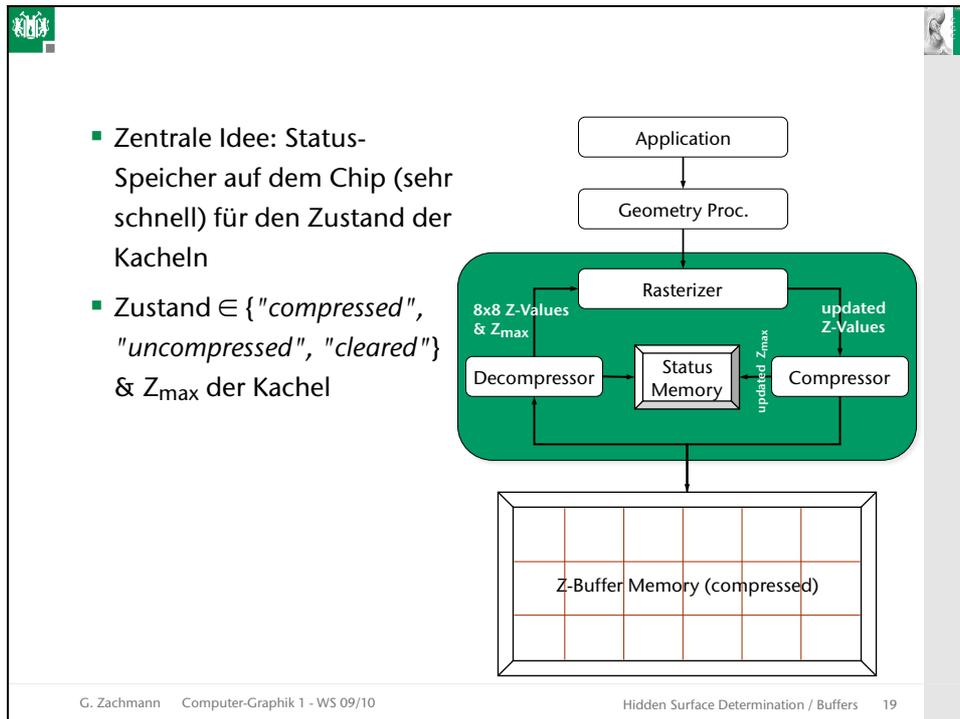
- Definition **Over-Drawing** = Maß dafür, wie oft ein Pixel tatsächlich überschrieben wird

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 17

## Implementierung in aktueller Graphik-Hardware

- Problem: Bandbreite zwischen Rasterizer und Speicher beträgt ca. 18 Gbyte/sec!
  - Annahmen: Auflösung 1280x1024, 4x depth complexity pro Pixel, pro Fragment: 1x Z-Buffer-Read + 1x Z-Buffer-Write + 1x Color-Buffer-Write + 2x Texture-Read, pro Read/Write 32 Bit
  - Aktueller Speicher erlaubt ca. 10 Gbyte/sec [2002]
- Wie implementiert man schnell `glClear (DEPTH_BUFFER_BIT)`?
- Wie implementiert man den HZB?
- Lösung: Z-Buffer in **Kacheln** aufteilen und **komprimieren**

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 18



- Z-Buffer lesen:
  - Decompressor liest zuerst  $Z_{\max}$  aus dem Status Memory
  - Verschiedene Tests möglich:
    - Teste die Z-Werte der 4 Ecken der Kachel gegen  $Z_{\max}$ 
      - Bemerkung: diese 4 Z-Werte kann man in den Nachbarkacheln wiederverwenden
    - Teste die Z-Werte der 3 Ecken des Dreiecks gegen  $Z_{\max}$
    - Berechne alle Z-Werte der Pixel in der Kachel und teste gegen  $Z_{\max}$
  - Fordere Z-Werte der Kachel aus dem Z-Buffer an, falls Test "fehlschlägt"
  - Falls Status der Kachel = "compressed", dekomprimiere Z-Werte vor der Weiterleitung an den Rasterizer
- Nennt sich "early z exit" oder "HyperZ" bei den Graphikkartenherstellern
- Kompression (z.B. DPCM) erreicht bis zu Faktor 4:1

### Performance-Gewinn in einer Graphikkarte

- Beispiel: [ATI RADEON 9700 PRO](#) [2003]
  - 3 Levels: 1. 8x8 Z-Block, 2. 4x4-Block, 3. "Early Z"-Test
- Performance-Gewinn:

**Scene Order Rendering Performance - 8 Layers 1280x1024**

Rendering Method	Performance (Units)
Front-to-Back	1754.99
Back-to-Front	244.99
Random	704.25

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 22

## Object-Space vs. Image-Space

- **Image Space Algorithmus:** arbeitet im diskreten(!) 2D-Bildraum
  - Hier: bestimme für jeden Pixel, welches Objekt sichtbar ist
  - Funktioniert auch bei dynamischen Szenen, da i.A. wenig / keine Hilfsdatenstrukturen
  - Beispiel: Z-Buffer, hierarchischer Z-Buffer
- **Object Space Algorithmus:** ganz allg Algorithmen, die direkt auf den 3D-Koord. der Objekte arbeiten (mit Floating-Point)
  - Hier: bestimme vor dem Abschicken von OpenGL-Befehlen, welche Objekte/Polygone andere verdecken
  - Berechnung basiert oft auf den Aufbau komplexer Hilfsdatenstrukturen
  - Funktioniert besser bei statischen Szenen

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 23

## Binary Space Partition (BSP) Tree [ca. 1982]

- Ein Object-Space-Algorithmus
- Rekursive Unterteilung des Raumes zur Tiefensortierung
- Sehr effizient für statische Szenen
- Ermöglicht sehr schnell Hidden-Surface-Elimination für alle Viewpoints mittels Painter's Algorithm
- Ursprünglich fürs Rendering ohne Z-Buffer entwickelt, heute immer noch eine sehr wichtige Datenstruktur in der CG
  - Wurde sogar 1996 noch im Spiel Quake (und auch Quake II?) verwendet für Hidden-Surface-Elimination!  
([http://en.wikipedia.org/wiki/Quake\\_engine](http://en.wikipedia.org/wiki/Quake_engine))

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 24

## Grundlegende Idee

- Annahme (vorerst): keine 2 Polygone schneiden sich
- $F_p$  sei die implizite Gleichung der Ebenengleichung die das Polygon  $p$  enthält
- Ein Hidden-Surface-Algo für folgende Szene:
 

```

      if  $F_{t1}(\text{eye}) > 0$  :
        draw  $t_2$ 
        draw  $t_1$ 
        draw  $t_0$ 
      else:
        draw  $t_0$ 
        draw  $t_1$ 
        draw  $t_2$ 
      
```

- Funktioniert für beliebigen Viewpoint!

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 25

## Rendering-Algorithmus mit BSPs

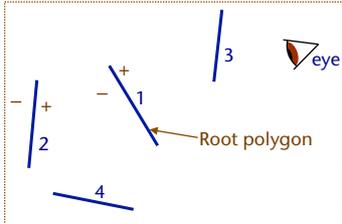
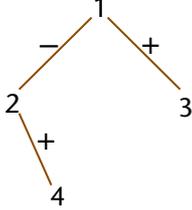
- BSP Tree: unterteilt Raum rekursiv in positive und negative Teile
  - Knoten = Zeiger auf Polygon(e) und Ebenengleichung, in der diese Polygone liegen
  - Linker / rechter Sohn = Unterbaum der all Polygone enthält, die im negativen / positiven Halbraum der Ebene liegen
  - Polygone, die auf beiden Seiten liegen, werden gesplittet
- Rendering (von hinten nach vorne):
  - Beginne bei der Wurzel
  - Zeichne Polygone rekursiv auf der Gegenseite vom Viewpoint
  - Zeichne Polygon(e) im Knoten
  - Zeichne rekursiv Polygone auf der selben Seite wie Viewpoint

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 26

```

draw(node, eye):
  if node.empty():
    return;
  if node.plane(eye) < 0 :
    draw( node.plus, eye )
    rasterize( node.triangle )
    draw( node.minus, eye )
  else
    draw( node.minus, eye )
    rasterize( node.triangle )
    draw( node.plus, eye )

```

Reihenfolge beim Zeichnen: 2, 4, 1, 3

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 27

## Aufbau eines BSP Tree

1. Wähle ein Polygon und setze es als Wurzelement
2. Partitioniere die Menge der restlichen Polygone in zwei Teilmengen, je nachdem auf welcher Seite sie liegen
3. Schneidet ein Polygon die Ebene, dann unterteile es in zwei Polygone, jeweils ein Teil auf einer Seite
4. Baue rekursiv je einen BSP für alle Polygone auf der negativen bzw. positiven Seite und hänge diese als Kinder an die Wurzel
5. Stoppe wenn ein Unterbaum nur noch ein Polygon enthält

- NB: diese Art BSP heißt **Auto-Partition**

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 28

Beispiel

- Ausgangsszene:

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 29

- Wähle z.B. Polygon 3 als Wurzelement

```

graph TD
    3((3)) --- front[1<br/>2<br/>4a]
    3 --- back[4b<br/>5]
    
```

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 30

Wähle Polygon 2 und 4b als nächste Knoten

```

    graph TD
      3((3)) --- front[front]
      3 --- back[back]
      front --- 2((2))
      back --- 4b((4b))
      2 --- 1b[1b]
      2 --- 4a[4a]
      2 --- 1a((1a))
      4b --- 5((5))
      style 1b fill:#fff,stroke:#000
      style 4a fill:#fff,stroke:#000
      style 1a fill:#fff,stroke:#000
      style 5 fill:#fff,stroke:#000
      style 2 fill:#fff,stroke:#000
      style 4b fill:#fff,stroke:#000
      style 3 fill:#fff,stroke:#000
      style 1b,4a stroke:#000,stroke-width:2px
  
```

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 31

Nun wähle Polygon 1b als Knoten

```

    graph TD
      3((3)) --- front[front]
      3 --- back[back]
      front --- 2((2))
      back --- 4b((4b))
      2 --- 1b((1b))
      2 --- 1a((1a))
      1b --- 4a((4a))
      4b --- 5((5))
      style 1b fill:#fff,stroke:#000
      style 1a fill:#fff,stroke:#000
      style 4a fill:#fff,stroke:#000
      style 5 fill:#fff,stroke:#000
      style 2 fill:#fff,stroke:#000
      style 4b fill:#fff,stroke:#000
      style 3 fill:#fff,stroke:#000
      style 1b,1a,4a,5 stroke:#000,stroke-width:2px
  
```

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 32

### Beispiel fürs Rendering mittels BSP

- Angenommen, der Viewpoint befindet sich wie hier dargestellt

```

    graph TD
      3((3)) -- front --> 2((2))
      3 -- back --> 4b((4b))
      2 --> 1b((1b))
      2 --> 1a((1a))
      1a --> 4a((4a))
      4b --> 5((5))
      eye((eye))
  
```

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 33

- Viewpoint liegt auf der Rückseite von 4b, somit zeichnen wir zuerst die Polygone auf der Vorderseite von 4b

- Zeichne 4b und im Anschluss die Polygone auf der Rückseite von 4b, also 5

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 34

▪ Nun zeichnen wir die Vorderseite von 3

front back

3

2

4b

1b

1a

5

4a

eye

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 35

▪ Der Viewpoint liegt auf der Rückseite von 2, somit werden erst die Polygone auf der Vorderseite von 2 gezeichnet

front back

3

2

4b

1b

1a

5

4a

eye

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 36

Der Viewpoint liegt auf der Rückseite von 1b, somit werden erst die Polygone auf der Vorderseite von 1b gezeichnet

Zeichne 1b, danach die Polygone auf der Rückseite von 1b, also 4a

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 37

Danach zeichne 2

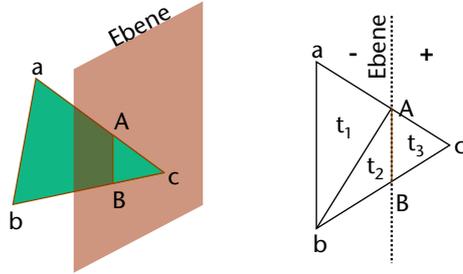
Im Anschluss die Polygone auf der Rückseite von 2, also 1a

Ergibt Gesamtreihenfolge: 4b, 5, 3, 1b, 4a, 2, 1a

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 38

## Zerschneiden von Dreiecken

- Dreieck schneidet die Ebene → unterteilen



$$\begin{aligned} t_1 &= (a, b, A) \\ t_2 &= (b, B, A) \\ t_3 &= (A, B, c) \end{aligned}$$

- Achtung: Reihenfolge der Eckpunkte muß beibehalten werden, damit sich Normale nicht ändert
- Angenommen c liegt allein auf einer Seite der Ebene und es gilt  $f_{\text{plane}}(c) > 0$ , dann:
  - Füge  $t_1$  und  $t_2$  in den negativen Unterbaum ein
  - Füge  $t_3$  in den positiven Unterbaum ein

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 40

## Wie bestimmt man A und B ?

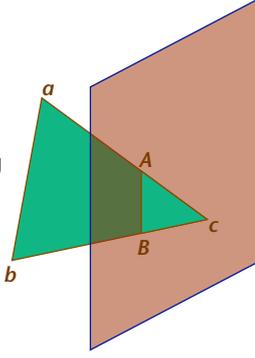
- A: Schnittpunkt der Gerade zwischen a und c und der Ebene  $f_{\text{plane}}$
- Verwende Parameterform der Geradengleichung  $p(t) = a + t(c - a)$
- Setze p in die Ebenengleichung ein

$$\begin{aligned} f_{\text{plane}}(p) &= (n \cdot p) - d \\ &= n \cdot (a + t(c - a)) - d \stackrel{!}{=} 0 \end{aligned}$$

- Berechne t und setze es in p(t) ein, um A zu berechnen

$$t = \frac{d - (n \cdot a)}{n \cdot (c - a)}$$

- Wiederhole dies zur Berechnung von B



G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 41

## Demo

Quelle: Paton J. Lewis - <http://symbolcraft.com/graphics/bsp/>

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 42

## Zusammenfassung

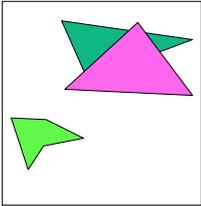
Vorteile	Nachteile
<ul style="list-style-type: none"> <li>▪ Sehr effiziente Datenstruktur um Polygone bzgl. eines Punktes zu sortieren!</li> <li>▪ Unabhängig vom Viewpoint</li> <li>▪ Wird auch für andere Aufgaben benötigt</li> </ul>	<ul style="list-style-type: none"> <li>▪ Viele kleine Polygone (wg. Splitting)</li> <li>▪ Starkes Over-drawing           <ul style="list-style-type: none"> <li>▪ Viele Pixel werden „umsonst“ geschrieben (wg. Back-to-front-Sortierung)</li> </ul> </li> <li>▪ Schwierig, den Baum ausgeglichen zu halten</li> </ul>

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 43

## Warnock's Algorithmus [1996]



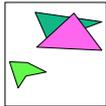
- Ein Image-Space-Verfahren, das auf einer rekursiven Unterteilung des Bildschirms beruht, bis die einzelnen Gebiete "homogen" sind
- Heute nicht mehr relevant (im Moment)
- Zeigt aber sehr schön folgendes algorithmisches Prinzip:
  - Kann man eine geometrische Entscheidung nicht für den ganzen Bereich fällen, so teile diesen erst einmal auf (hier: Bildraum wird aufgeteilt)
  - Ist im Prinzip eine Variante von Divide-and-Conquer

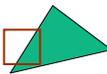
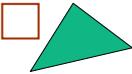


G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 44

## Idee

- Unterteile den Bereich in 4 gleiche Gebiete
- Treffe für jedes Teilgebiet die Entscheidung, welches Polygon (vorne) gezeichnet werden soll
- In jedem Gebiet liegt einer der folgenden 3 Fälle vor:
  1. Das Gebiet liegt komplett innerhalb eines Polygons, und dieses befindet sich vor allen anderen Polygonen, die auch innerhalb dieses Gebietes liegen
  2. Genau 1 Polygon liegt im Gebiet (entweder vollständig innerhalb des Gebiets oder dieses schneidend)
  3. Gebiet und Polygon sind disjunkt



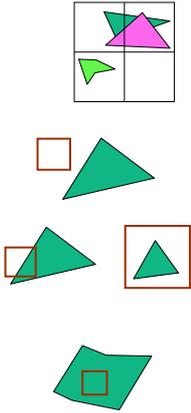



G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 45

- Nicht vom Gebiet geschnittene Polygone beeinflussen das Gebiet nicht
- Schneidet ein Polygon das Gebiet, so beeinflusst der außerhalb liegende Teil das Gebiet nicht
- In jedem Schritt berechnen wir die Farbe des Gebietes; ist die Berechnung nicht eindeutig, dann unterteile es erneut

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 46

- Der Algo unterteilt nun rekursiv den Bildschirm (und die Menge der Polygone)
- Bei jeder Rekursion wird das Teilgebiet untersucht:
  1. Kein Polygon innerhalb des Gebietes → fülle mit der Hintergrundfarbe
  2. Nur 1 Polygon liegt ganz oder teilweise innerhalb des Gebiets → fülle Gebiet mit der Hintergrundfarbe und zeichne anschließend den Teil des Polygons, der innerhalb liegt
  3. Wird das Gebiet von genau 1 Polygon umschlossen (kein Schnitt mit einem anderen Polygon) → färbe Gebiet komplett mit der Farbe des Polygons
  4. Umschließt, schneidet oder enthält das Gebiet mehr als 1 Polygon, aber ein Polygon liegt vor allen anderen → fülle das Gebiet mit der Farbe dieses Polygons
    - Anderenfalls: unterteile das Gebiet und fahre mit Rekursion fort

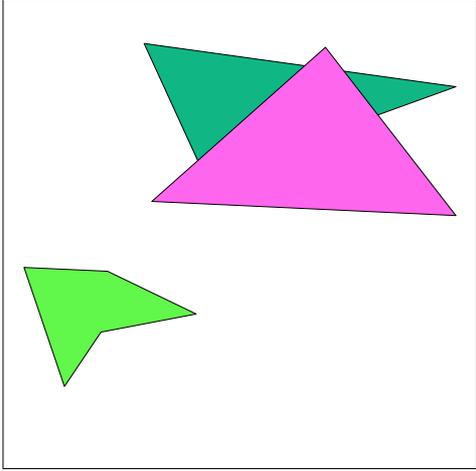


G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 47

- Unterteilung wird fortgeführt bis:
  - Alle Gebiete entsprechen einer der vier Kriterien
  - Die Größe des Gebietes entspricht einem Pixel
    - In diesem Fall wird die Farbe irgendeines Polygons zum Füllen gewählt; oder ...
    - Man füllt mit dem Mittelwert aller Polygonfarben; oder ...
    - Man macht Anti-Aliasing zwischen den Polygonen

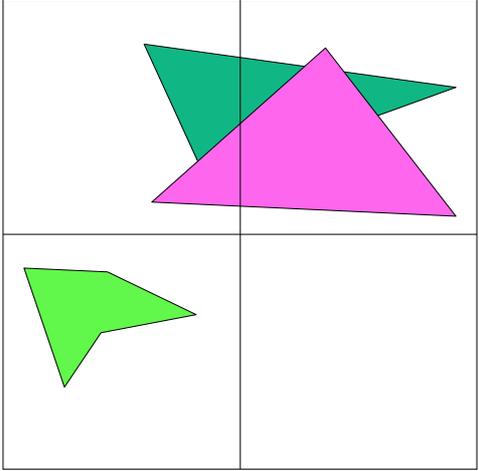
G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 48

### Beispiel



Ausgangsszene

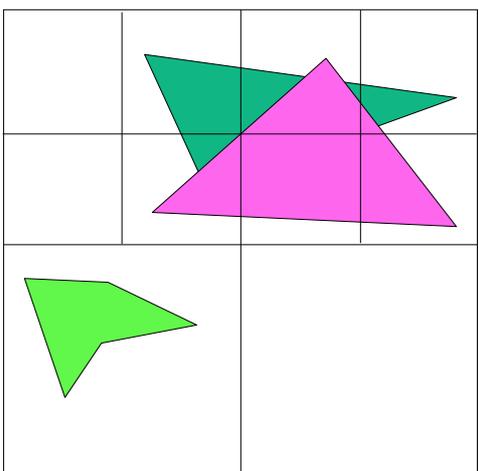
G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 49



Erste Unterteilung

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 50

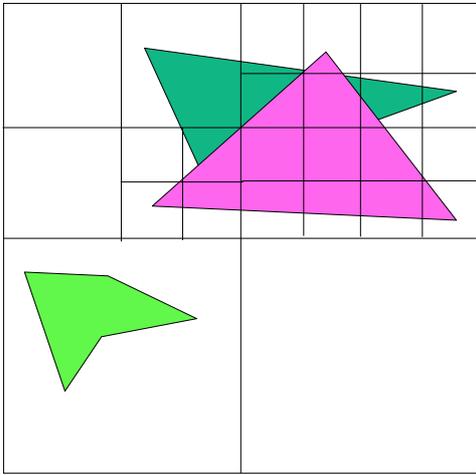
The diagram illustrates the first step of a hidden surface algorithm. It features a 2x2 grid. In the top-left quadrant, a cyan triangle is partially visible behind a pink triangle. In the bottom-left quadrant, a green triangle is visible. The rest of the grid is empty. The text 'Erste Unterteilung' (First Subdivision) is centered below the grid.



zweite Unterteilung

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 51

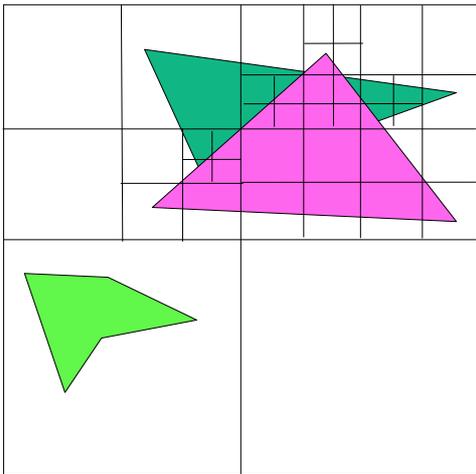
The diagram illustrates the second step of the hidden surface algorithm. It features a 4x2 grid. The cyan and pink triangles from the first step are now subdivided into smaller regions by the additional grid lines. The green triangle remains in the bottom-left quadrant. The text 'zweite Unterteilung' (Second Subdivision) is centered below the grid.



dritte Unterteilung

G. Zachmann Computer-Graphik 1 - WS 09/10

Hidden Surface Determination / Buffers 52



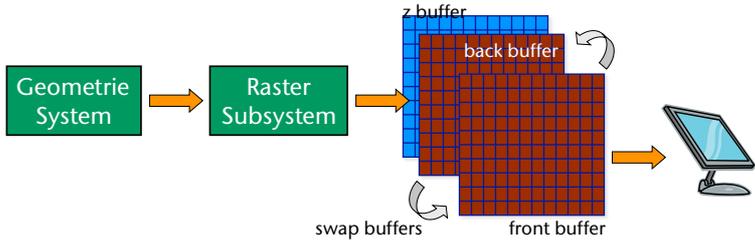
vierte Unterteilung

G. Zachmann Computer-Graphik 1 - WS 09/10

Hidden Surface Determination / Buffers 53

## Speaking of "Buffers"

- Es gibt noch viele weitere Buffer in einem Framebuffer
- Der **Double-Buffer**:
  - Problem beim **Single-Buffering**: **Flickering**
  - Lösung: 2 Buffers
  - Front Buffer** = Color-Buffer, der vom Display gerade angezeigt wird
  - Back Buffer** = Color-Buffer, in den gerade gezeichnet wird

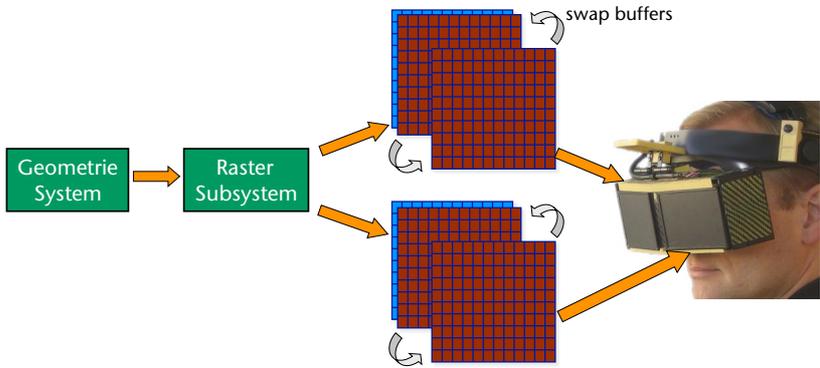


The diagram illustrates the double-buffering process. It starts with a 'Geometrie System' (Geometry System) which feeds into a 'Raster Subsystem' (Raster Subsystem). The Raster Subsystem outputs to a 'z buffer' (Z-buffer) and a 'back buffer' (Back Buffer). The back buffer is then swapped with the 'front buffer' (Front Buffer), which is then displayed on a monitor. The swap buffers process is indicated by curved arrows between the back and front buffers.

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 54

## Quad Buffers

- Für Stereo- (3D-) Rendering muß man 2 unterschiedliche Bilder generieren: je eines für das linke bzw. rechte Auge
- Lösung: 2 Front buffers, 2 back buffers (und 2 Z-Buffer!)



The diagram illustrates the quad-buffering process for stereo rendering. It starts with a 'Geometrie System' (Geometry System) which feeds into a 'Raster Subsystem' (Raster Subsystem). The Raster Subsystem outputs to two sets of buffers: a 'z buffer' and a 'back buffer' for each eye. The back buffers are then swapped with the front buffers, which are then displayed on a VR headset. The swap buffers process is indicated by curved arrows between the back and front buffers for each eye.

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 55

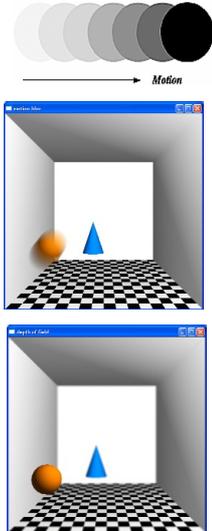
## Accumulation Buffer (A-Buffer)

- Wie der Color Buffer, kann aber aufaddieren (akkumulieren), und am Ende durch Anzahl der Frames teilen
- Funktionsweise:
  - Szene in den normalen Color-Buffer rendern
  - Mit `glAccum(GL_ACCUM, 1/N)` wird Color-Buffer zum Accumulation-Buffer aufaddiert
  - Am Ende mit `glAccum(GL_RETURN)` den Acc.-Buffer in Color-Buffer übertragen

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 56

## Anwendungen

- Beispiel: **Full-Screen Anti-Aliasing**
  - Renderere Szene mehrfach von leicht verschiedenen Viewpoints
  - Akkumuliere alle auf und bilde Mittelwert
- Beispiel: **Motion Blur**
  - Verwende `glAccum(GL_MULT, decayFactor)` mit  $decayFactor \in [0,1]$
  - Render Szene mehrfach mit immer niedrigerem `decayFactor`
- Beispiel: **depth-of-field** (Tiefenunschärfe)
  - Renderere Szene mehrfach mit verschiedenen Blickrichtungen **und** verschiedenen Kamerapos. (nur leicht variiert! = *jittering*), so daß immer derselbe Fokus in der Mitte des Bildschirms liegt



G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 57




- Achtung: der Accumulation Buffer ist sehr teuer
  - Platz und Zeit
- Für die meisten Anwendungen gibt es heute bessere (dafür etwas kompliziertere) Techniken
- Fazit: oft wird der Accumulation-Buffer nur noch in Software implementiert – oder gar nicht

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 58

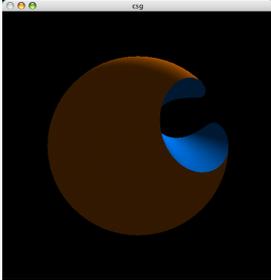



## Stencil Buffer

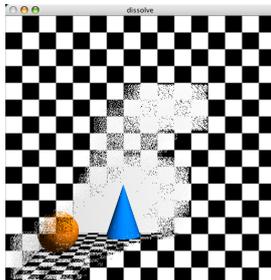
- Stencil-Buffer: eine Art "Vergleichs-Buffer"
  - Ähnlich zu Z-Buffer, aber mit anderen Features
- Stencil-Test:
  - Vergleiche vor dem Setzen eines Pixels den Referenzwert mit Stencil-Wert
  - Verändere ggf. den Wert im Stencil-Buffer
  - Je eine Operation für Stencil-Fail, Stencil-OK/Z-Fail, Stencil-OK/Z-Pass
- Einfaches Beispiel:
  - Zeichne Szene nur dort, wo ein bestimmtes Objekt A sichtbar ist
  - Alle Buffer inkl. Stencil-Buffer löschen
  - Objekt A rendern, dabei Stencil-Buffer überall dort auf 1 setzen
  - Rest der Szene zeichnen, aber nur dort, wo Stencil-Wert = 1

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 59

- Beispiel: CSG-Operationen (Schnitt, Differenz, ...)



- Beispiel: "Dissolve"



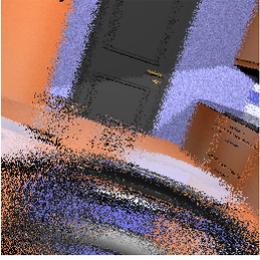
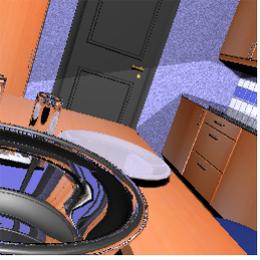
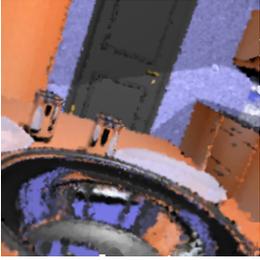
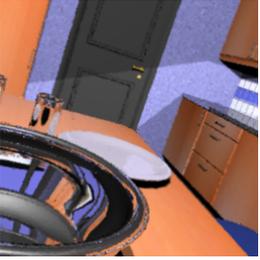
G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 60

## Frameless Rendering [1994, 2005]

- Annahme: die Anzahl der Pixel im Frame ist der bestimmende Faktor für die Rendering-Zeit (→ *"fill limited"*)
  - Z.B. der Fall bei wenigen Polygonen und großem Display; oder bei Ray-Tracing (später)
- Idee: verwende das alte Frame wieder, und erneuere nur einige, zufällig ausgewählte Pixel
  - Konsequenz: es gibt keinen Double-Buffer mehr
  - Wenn die Szene dann statisch wird, werden sukzessive alle Pixel erneuert, und das Bild konvergiert zum "klassisch" gerenderten Bild
- Vorteil: wesentlich geringere Latenz zwischen Kamera-Bewegung und Erscheinen eines neuen Frames auf dem Display

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 63

Beispiele

	dynamic scene	static scene
Einfaches Frameless Rendering		
Temporally Adaptive Reconstruction		

G. Zachmann Computer-Graphik 1 - WS 09/10 Hidden Surface Determination / Buffers 64