

Toni Lama

3D-Welten

PDF-Datei zum Buch

Palisadenwand - Tutorial

Hallo und herzlich Willkommen zur Palisadenwand-Übung. Dieses Tutorial soll Ihnen ein tieferes Verständnis der **#while**-Schleifen und der Zufalls-(Zahlen-)Konstruktion ermöglichen – hoffe ich ☺.

Voraussetzungen:

- POV-Ray for Windows 3.5
- ein grundlegendes Verständnis der **#while** – Schleifen sowie
- der Zufallszahlen (beides in Kapitel 11 erläutert)

Vorbereitende Arbeiten:

- wir öffnen eine leere Szene (**File – New File**)
- fügen das Checkered floor template ein (**Insert – Scene templates – Checkered floor**)
- bringen die **plane** auf **y, 0**
- und die **camera** auf **location <0.0, 1.5, -4.0>** und **look_at <0.0, 1.0, 0.0>**
- entfernen die Kugel (das gesamte **sphere – item** löschen)

Somit ist das unsere Ausgangssituation:



Damit wir später nicht in den *items* herumfuschen legen wir die Texturen außerhalb fest:

```
#declare stamm = pigment {color Red}
#declare spitze = pigment {color Green}
#declare schlaufe = pigment {color Yellow}
```

Nun konstruieren wir den einzelnen Pfahl:

```
union{
cylinder { 0*y, 2*y, 0.1 texture {stamm} }
cone { 2*y, 0.1, 2.25*y, 0.0 texture {spitze} }
} // union
```

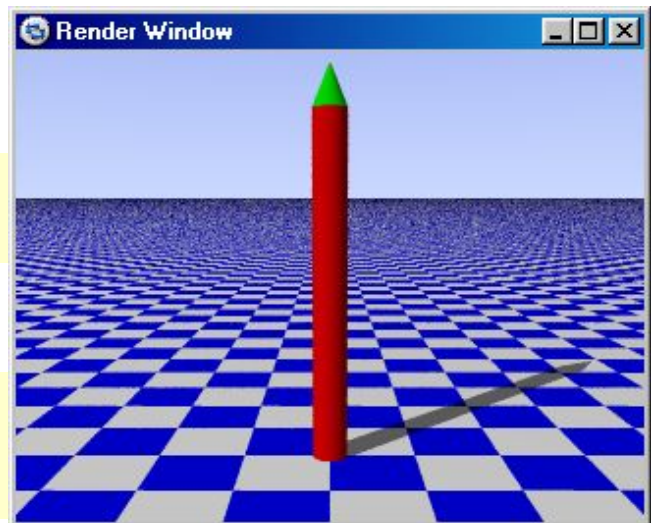


Bild 2b: palisade_1.pov

Das ist unser Hauptdarsteller:

Unsere Palisadenwand soll aus 20 Stämmen bestehen, deshalb wickeln wir eine Schleife um die **union**. Über der **union** schreiben wir:

```
#declare ticker = 1;
```

```
#while(ticker < 21)
```

und unter der **union** fügen wir hinzu:

```
#declare ticker = ticker+1;
```

```
#end
```

Das reicht natürlich nicht, denn so würde der Pfahl zwar 20 mal, aber immer an der gleichen Stelle gesetzt. Deshalb ergänzen wir das **item** um:

```
translate (-2.3*x)+(ticker*0.2*x)
```

vor der schließenden Klammer.

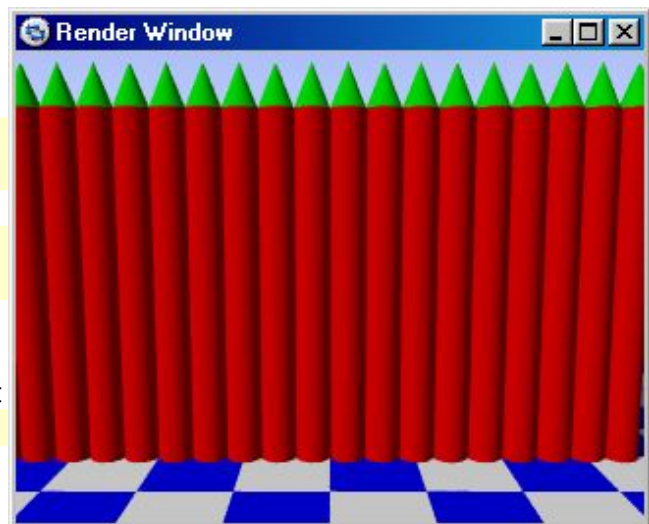


Bild 2c: palisade_2.pov

Das Problem ist nicht zu leugnen: so sehen vielleicht 20 Bleistifte aus industrieller Fertigung aus, aber niemals eine echte Palisadenwand. Mehr Realismus bitte! *OK then*, bedienen wir uns des Zufalls: dazu geben wir außerhalb der Schleife eine Zahlenkette in Auftrag mit

```
#declare zufall = seed(3);
```

Der Startwert 3 wurde von mir willkürlich gewählt; Sie wissen, daß es auch jede andere Zahl sein kann. Die Vereinbarung muß außerhalb der Schleife stehen, andernfalls würde die Zahlenkette bei jedem Durchgang neu beginnen und **rand**(zufall) hätte immer den gleichen Wert – dann können wir uns den Aufwand gleich sparen.

Nun, da die Zahlenkette definiert ist, können wir bei jedem Durchgang einen neuen Durchmesser festlegen. Wir machen das innerhalb der Schleife ganz elegant mit

```
#declare durchmesser = rand(zufall);
```

und ändern die **items** wie folgt:

```
cylinder { 0*y, 2*y, 0.1*durchmesser usw.
```

```
cone { 2*y, 0.1*durchmesser usw.
```

Die Überlegung scheint logisch – da **rand**(zufall) höchstens 1 sein kann, kann ein Pfahl höchstens $1 * 0.1 = 0.1$ im Radius betragen, die Pfähle können also nicht überlappen.

Das Ergebnis ist trotzdem Mist, wie Sie sehen:

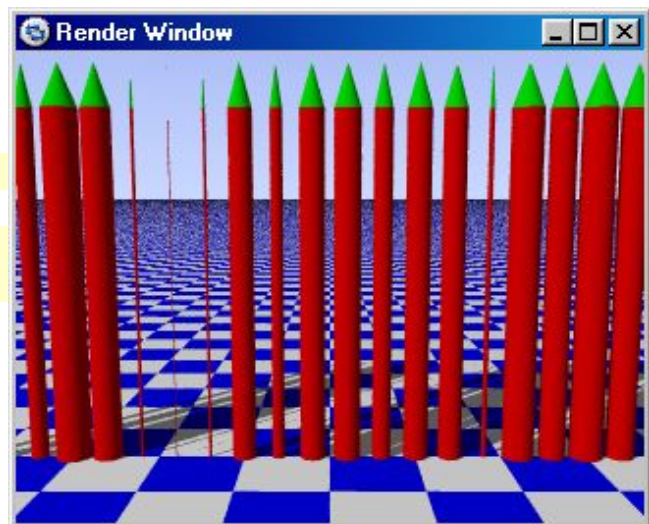


Bild 3a: palisade_3.pov

Wir haben übersehen, daß **rand**(zufall) auch mal Null betragen oder nahe bei Null liegen kann. Dann bekommen wir bestenfalls einen dünnen Faden oder gar nichts (Null mal irgendwas ist immer noch Null). Die Variante mit „minus durchmesser“ bringt uns auch nicht weiter, wovon Sie sich schnell selbst überzeugen können.

Wie also dann? Unser Ziel sind Pfähle, die maximal einen Radius von 0,1 haben, aber auch leicht darunter liegen können. *Leicht* – sonst sieht es aus wie im Bild 3a. Deshalb gilt es, vom definierten Radius 0.1 einen kleinen Betrag abzuziehen. Da wir nicht wissen, ob `rand()` 0.001 oder 0.999 liefern wird, gehen wir auf Nummer Sicher und geben uns mit einer Abweichung von einem Prozent zufrieden:

```
#declare durchmesser = (rand(zufall) / 100);
```

und in die *items* setzen wir:

```
-durchmesser
```

(minus durchmesser). So sollte es aussehen: die unterschiedlichen Durchmesser lassen nicht überall Lücken, das sieht zufällig aus.

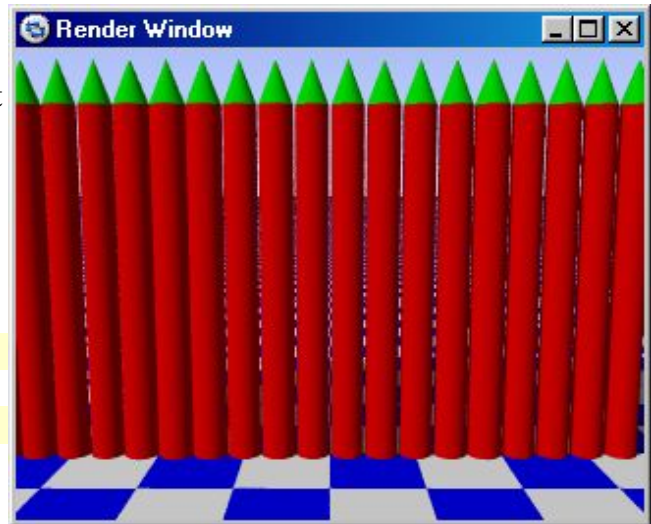


Bild 3b: palisade_4.pov

Bei einer richtigen Palisadenwand sind die Pfähle vertäut, also benötigen wir oben und unten Schlaufen. Die machen wir mit einem `torus`, zunächst sehr gleichmäßig innerhalb des *loops*.

Wir bilden zwei Reihen:

```
torus { 0.1, 0.01 texture {schlaufe }
scale 0.5*y
translate 0.4*y }
```

und

```
torus { 0.1, 0.01 texture {schlaufe }
scale 0.5*y
translate 1.8*y }
```

das sieht ebenfalls zu gleichmäßig aus, deshalb ...

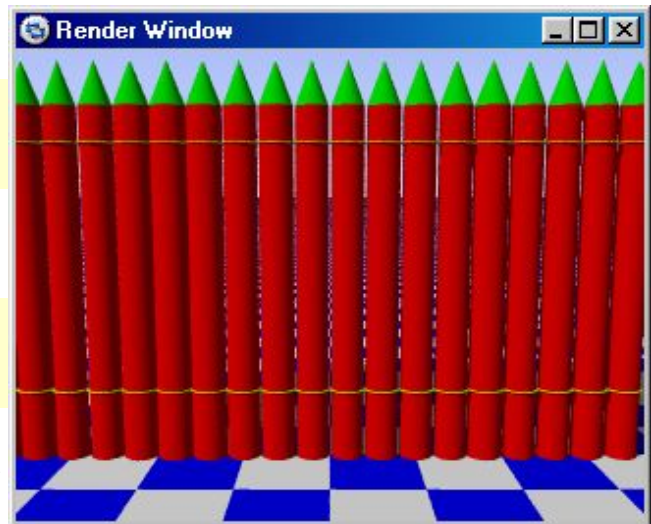


Bild 4a: palisade_5.pov

... drehen wir die Schlaufen ein wenig um die x-Achse:

```
torus { 0.1, 0.01 texture {schlaufe }
scale 0.5*y
rotate (-durchmesser*600)*x
translate 0.4*y }
torus { 0.1, 0.01 texture {schlaufe }
scale 0.5*y
rotate (durchmesser*600)*x
translate 1.8*y }
```

Bedingt durch die Kameraperspektive sieht man den Unterschied hier deutlicher in der oberen Reihe, rendern Sie das Bild ruhig größer.

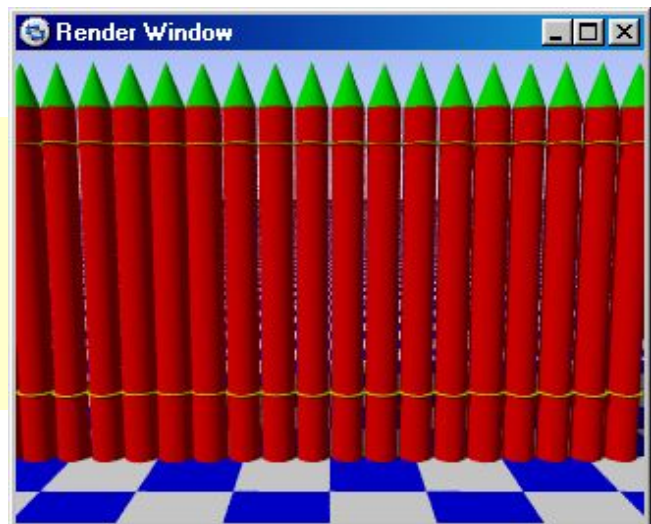


Bild 4b: palisade_6.pov

Was macht der irrsinnig hohe Wert von 600 in der **rotate**-Anweisung? Nun, grundsätzlich haben wir an dieser Stelle zwei Möglichkeiten: entweder wir rufen aus der Zahlenkette den nächsten Wert ab (dann verschiebt sich die Palisade, weil der Zufallswert des zweiten Pfahls für die Schlaufen verbraucht wurde) oder wir benutzen für alle Operationen innerhalb eines Durchgangs den selben Zufallswert, der dann für das entsprechende *item* umgerechnet werden muß. Ich habe hier die zweite Möglichkeit gewählt. **durchmesser** hat ja eine Größe von (rand(zufall) durch hundert), also höchstens 0.01 – für die Pfähle OK, aber eine Drehung um 0.01 Grad oder weniger würden unsere Augen nicht erkennen. Deshalb wird der Wert hier mit 600 multipliziert, woraus sich eine maximale Drehung von 6 Grad ergibt. Klar, oder?

Ganz pffiffige Naturen könnten der Idee verfallen, die Tori auch noch um z zu drehen, das wird aber nix. Dann ist nämlich nicht mehr gewährleistet, daß die Schlaufen zwischen den Pfählen aneinanderliegen – das ist aber Bedingung, wenn alle Pfähle durch ein Seil verbunden sind.

Für den nächsten Gang benötigen wir ein *include-file*, also fügen wir oben ein: **#include "woods.inc"**

Nun wandeln wir die roten Säulen in echte Stämme um. Im ersten Schritt ändern wir die einfache rote Farbe in

```
texture {T_Wood2
rotate x*90
normal {dents -30 scale 0.1}
finish {crand 0.2 }
scale 3*y }
```

Danach ändern wir auch die „spitze“-Deklaration in

```
texture {T_Wood31
rotate x*90
normal {crackle 3 scale 0.1} }
```

und die Schlaufe in

```
texture {T_Wood23 rotate x*90 }
```

Wir rendern und bekommen ...



Bild 5: palisade_7.pov

Nun ja, das ist schon näher an dem, was man sich unter einer Palisadenwand so vorstellt, aber, ähm, seien wir mal ehrlich: *schlecht* ist noch eine freundliche Umschreibung.

Die Textur ist bei allen Pfählen identisch und bedarf dringend einer Überarbeitung. Das überlassen wir dem Zufall. Dazu fügen wir in den *items* Folgendes ein: **scale durchmesser*30**.

Noch vor dem Rendern mag die Frage auftauchen, warum wir diese **scale**-Anweisung ins *item* schreiben und nicht in die Texturdeklaration. Ganz einfach: unsere höchst unprofessionelle Arbeitsweise hat dazu geführt, daß im *scene file* erst die **#declares** der Texturen auftauchen und dann erst die Vereinbarung der Zufallskette (ich wette, Sie haben die Zeile direkt oberhalb des *loops* eingefügt). Wenn nun beim *parsing* im „stamm“ ein „durchmesser“ genannt wird, der erst Zeilen tiefer vereinbart wird, bekommt der Programmablauf einen Koller. *Daran hätten wir vorher denken müssen.*

Nun ist es egal; wir fügen das **scale** in den Texturaufruf des *items* ein. Die Palisade ändert sich deutlich:

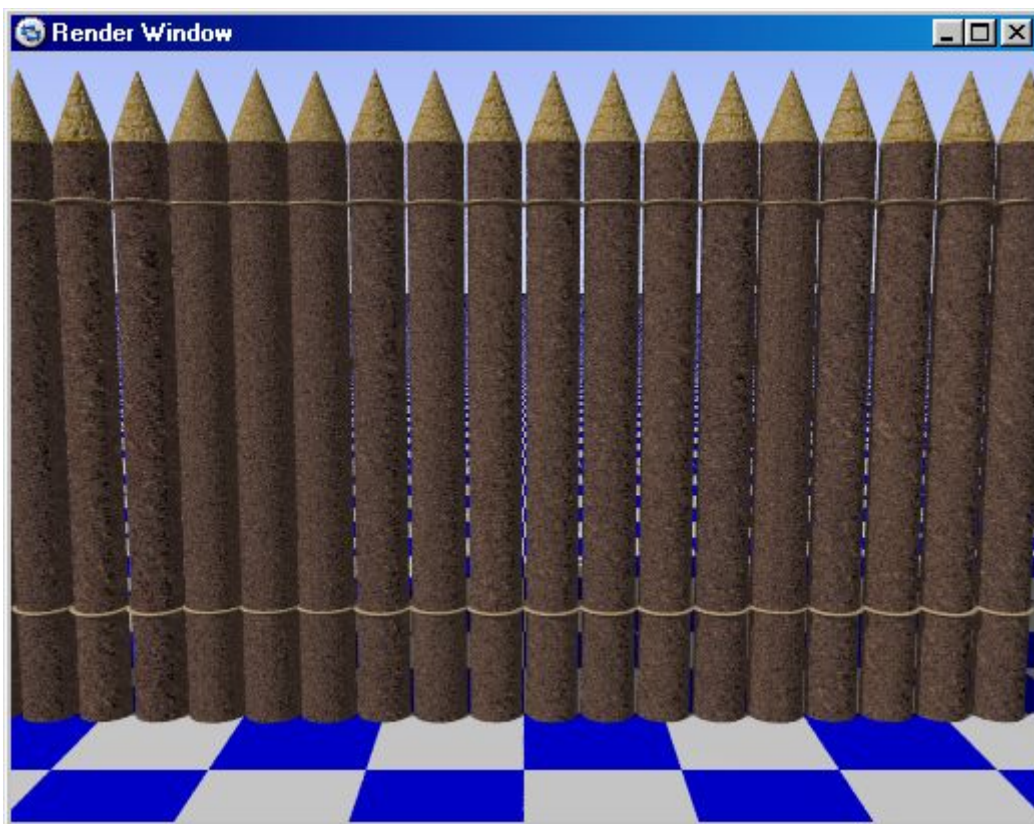


Bild 6: palisade_8.pov

OK, dabei wollen wir es belassen. Auch hier gibt es noch vieles zu verbessern, so könnte man etwa jeden Pfahl zufällig ein wenig in z versetzen, auch sind die Texturen nicht das Gelbe vom Ei, aber – Sie wollen ja nicht nur konsumieren, sondern auch selbst kreativ werden. Deshalb überlasse ich die Wand jetzt Ihrem Gusto. Spielen Sie ausgiebig mit ihr herum, um ein Gefühl für **rand()** zu entwickeln. Viel Spaß dabei und

so long!