Texturen, Nodedefinitionen

Texturen

Texturen liegen in Minetest als einzelne Bilddatei vor; können zu einem gewissen Grad auch im laufenden Spiel erzeugt werden

Je nach Verwendungszweck: Ganze Textur wird verwendet, oder verschiedene Ausschnitte davon (sogenanntes Spritesheet).

Verschiedene Dateiformate möglich. Hauptsächlich verwendet: png

Liegen in `textures`-Verzeichnis des Mods

Gemeinsamer Namensraum der Bilddateien aller Mods. Darum als Konvention: Im Dateinamen Modnamen als Prefix voranstellen: `modname_foo.png`

Texturkombinatoren

Texturen können miteinander kombiniert werden, um im laufenden Spiel neue Texturen zu erzeugen. Texturen mit Kombinator als String für beliebige Texturverwendungen angebbar

- Übereinander legen: `^`
- Texturtransformationen: `^[...`
  - drehen: `transformR90`
  - aufhellen: `brighten`
  - Nur unteren Anteil verwenden: `lowpart:prozentzahl:`
  - Verschobene Kopie: `combine`

Texturen können dabei auch geklammert werden, allerdings funktionieren nicht alle Kombinationen aus Transformationen mit Klammerung korrekt (ausprobieren)
Beispiel: `foo.png^(bar.png^buz.png)`

Zudem Texturkombinator für Itemcube, gedacht für Ansicht als Node im Inventar: `{...{...{...`

Texturverwendungen

Texturen können nahezu überall im Spiel verwendet werden

- Als Seiten von Nodes
- In Formularen
- Als HUD-Element (Dauerhafte Anzeige ohne Einfluss auf die Spielmechanik)
- Auf Modellen (Spieler, Spezielle Nodes, MOBs, . . .)
- Ansicht von Gegenständen im Inventar

Jede dieser Anwendungen kann durch Kombinatoren auf vielfältige Weise bereichert werden.

Verwendungen im Hauptspiel:
Erze auf Steinen, nur eine Steintextur, Erze getrennt gegeben
Feueranzeige im Ofen anteilig zu bisheriger Brenndauer

Modelle

Modelle werden an vielfältigen Stellen verwendet, um Objekte in beliebiger Form anzeigen zu können

- Spieler: Gibt ein Modell des Spielers im Grundspiel
- Spezielle Nodes: Neben Flüssigkeiten einzige Möglichkeit, um schiefe Teile an Nodes zu haben
- MOBs: Gibt verschiedene Mods mit Modellen für verschiedene Tiere

Modelle liegen im Verzeichnis `/models` des Mods, können in verschiedenen Dateiformaten vorliegen, das Grundspiel verwendet `.b3d` und `.obj`

Modelle können durch verschiedene Programme erzeugt werden, beispielsweise Blender; `.blend`-Dateien für viele Modelle im Spiel vorhanden, können aber nicht direkt geladen werden, Blender kann `.obj`-Dateien erzeugen

Spielermodell

Standardmodell für Spieler im Grundspiel

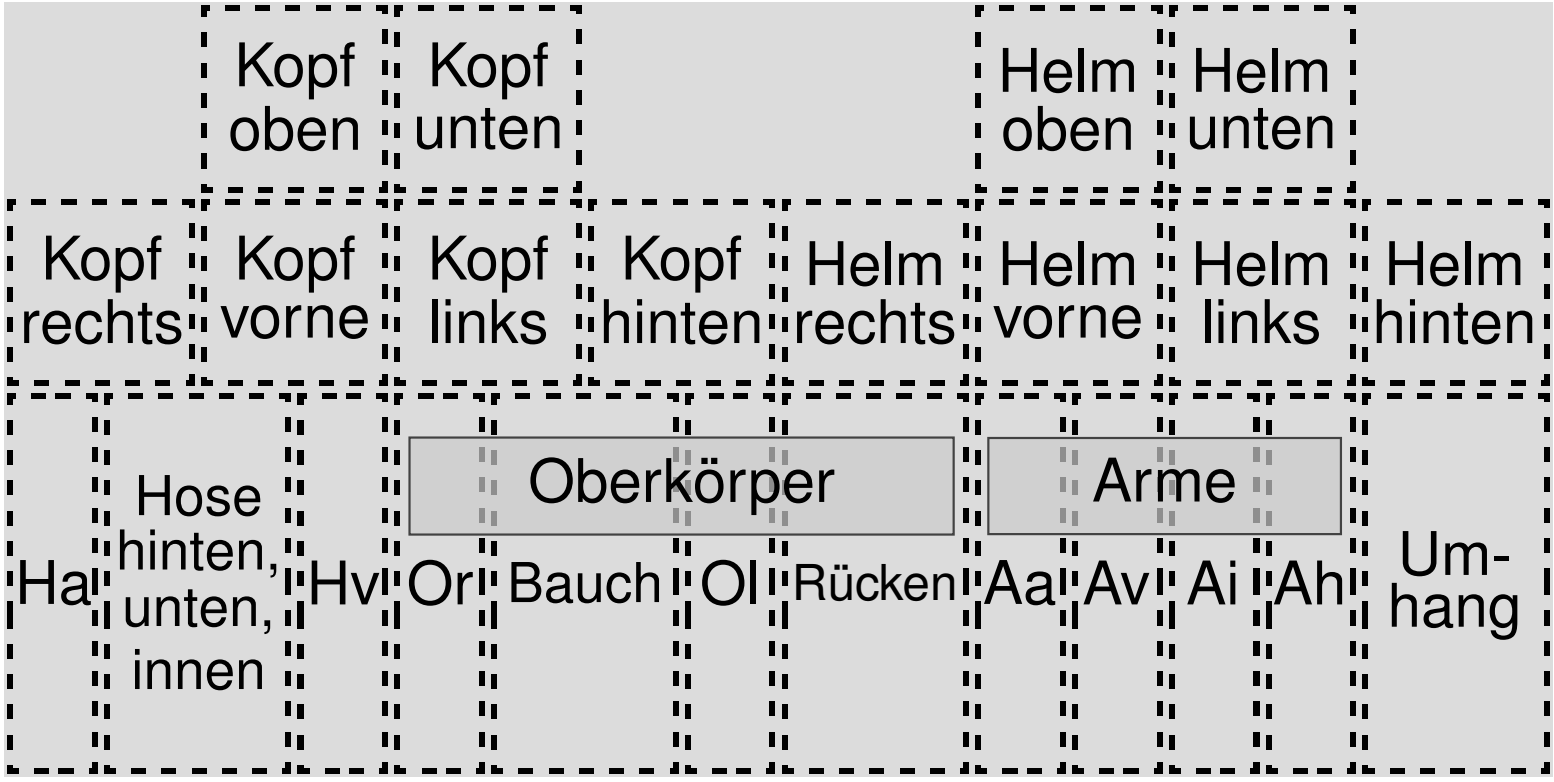
```
player:set_properties({
  visual = "mesh",
  textures = {texture},
  visual_size = {x=1, y=1},
})
```

`texture` ist eine beliebige Textur, auch eine mit Kombinatoren zusammengebaute Textur

`texture` ist ein Spritesheet, d.h. verschiedene Bereiche der Textur werden auf verschiedene Bereiche des Spielermodells gezeichnet

Spritesheet Spieler

Das Standardmodell für Spieler verwendet folgendes Spritesheet, insgesamt 64 $\times$ 32 Pixel



Helm: Zweite Schicht um den Kopf, knapp außerhalb

Ha Hose außen Hv Hose vorne Or Oberkörper rechts

Ol Oberkörper links Aa Arme außen Av Arme vorne und oben Ai Arme innen und unten Ah Arme hinten

Texturkombinatorbeispiel: Spielermodell

Spielermodell kann nur als ganzes ausgewechselt werden

Mit Texturkombinator leicht implementierbar: Teile des Spielermodells werden ausgetauscht

Anwendungsmöglichkeiten:

- Konfigurierbares Aussehen (Haare, Hautfarbe, ...)
- Anzeigbare Mimik
- Anzeige des Gesundheitszustands des Avatars (Schweiß bei geringer Energie, Mimik für Hunger, Wunden/Pflaster bei niedrigem Gesundheitszustand, ...)
- Anzeige der getragenen Ausrüstung am Avatar
- Auswechselbare Kleidung [Programmvorführung]

Noch mehr Darstellungsmöglichkeiten durch Kombination Modell und Textur, z.B. Werkzeug in der Hand anzeigen

Nodedarstellungen

Für die Nodes gibt es verschiedene Anzeigemodi, werden in der Nodedefinition festgelegt; Aussehen beeinflusst verhalten der Nodes nicht

- normal: Vollständige Node mit Texturen auf allen Seiten
- mesh: Frei festlegbares Modell
- nodebox: Aus koordinatenparallelen Boxen zusammengebaut
- airlike: Unsichtbar
- glasslike, glassframed, allfaces: Objekte dahinter werden auch angezeigt, darf darum transparente Teile enthalten
- liquid: kann teilweise durchsichtig sein
- plantlike: zwei gekreuzte Texturen in der Mitte
- firelike: Wie plantlike, hängt sich aber an Wände und Decken

Nodedarstellungen: Normal

```
minetest.register_node("...", {
  description = "Nodebeschreibungen",
  tiles = {"mod_foo0.png","mod_foo1.png"},
  groups = {choppy = 2, ...}
})
```

1 bis 6 verschiedene Texturen als Tiles angeben, das letzte angegebene Tile füllt die Tile-Liste auf

Reihenfolge der Texturpositionen: Oben, unten, vorne, hinten, links, rechts

Node Texturanimation

Animations-Table statt String für Textur

```
{ name = "mod_animateblock.png", backface_culling = false,
  animation = {
    type = "vertical_frames",
    aspect_w = 16,
    aspect_h = 16,
    length = 2.7,
  }, },
```

- name: Die Textur als Spritesheet der Animation
- backface_culling: Anzeige bei rückwärtiger Blickrichtung
- animation: Angaben zur Animation
  - type: In welcher Reihenfolge sind die Sprites angeordnet
  - aspect_w: Breite der Einzeltextur in Pixeln
  - aspect_h: Höhe der Einzeltextur in Pixeln
  - length: Zeitdauer komplette Animation in Sekunden

Node Texturanimation, weiteres

Texturanimationen lassen sich bei allen Darstellungstypen der Nodes verwenden

Als Wasseranimation im Hauptspiel verwendet

Wesentlich effizienter, als periodisch Nodes zu ersetzen, geschieht rein Clientseitig

Animation läuft immer als Schleife; Darstellungsschleifen für gleichen Nodetyp laufen synchron, werden manchmal unterbrochen und neu von vorne abgespielt

Für nicht synchrone Animation: Mehrere Nodes mit unterschiedlichen Animationen und unterschiedlicher Zeitdauer definieren, beim Setzen randomisieren, welche Node gesetzt wurde

Nodedarstellungen: Mesh

Genau so wie für Spieler kann ein Mesch für ein Node verwendet werden

```
minetest.register_node("...", {
  description = "Nodebeschreibungen",
  drawtype = "mesh",
  tiles = {"mod_foo0.png","mod_foo1.png"},
  mesh = "mod_thmesh.b3d",
  groups = {choppy = 2, ...}
})
```

Tiles sind hier die Materialien aus Sicht des Mesh, das Mesh kann die Texturen damit beliebig verwenden

Mesh ist die einzige Möglichkeit im Spiel um nicht-achsenparallele Teile an Nodes im Spiel zu haben

Nodedarstellungen: Nodebox

Nodebox kann einfachere Formen als ein Mesh erzeugen, hat aber immernoch große Formfreiheit

```
minetest.register_node("...", {
  description = "Nodebeschreibungen",
  drawtype = "nodebox",
  tiles = {"mod_foo0.png","mod_foo1.png"},
  node_box = {
    type = "fixed",
    fixed = {
      {-0.5,-0.125,-0.125,    0.5,0.125,0.125},
      {-0.125,-0.5,-0.125,    0.125,0.5,0.125},
      {-0.125,-0.125,-0.5,    0.125,0.125,0.5},
  },
  groups = {choppy = 2, ...}
})
```

Jede `fixed`-Zeile ist ein Quader in Koordinaten von -0.5 bis 0.5

Nodedarstellungen: Mesh vs. Nodebox

Nodebox ist wesentlich schneller zu erstellen: Direkt im Editor, keine sonstigen Programme und keine externen Dateien notwendig.

Mesh hat größere Gestaltungsfreiheit

Bei sehr komplexen Formen ist Mesh sogar schneller: Es werden nur die sichtbaren Flächen gezeichnet, wenn ein Teil den Rest verdeckt, so wird der Rest nicht gezeichnet; bei Nodebox werden alle (Vorderseiten) gezeichnet

Nodedarstellungen: Airlike

Durchsichtige Nodes

```
minetest.register_node("...", {
  description = "Nodebeschreibungen",
  drawtype = "airlike",
  tiles = {"mod_foo0.png","mod_foo1.png"},
  groups = {choppy = 2, ...}
})
```

Angegebene Texturen werden nicht verwendet

Erinnerung: Verhalten der Node wird durch Aussehen nicht beeinflusst
$\Rightarrow$ Ermöglicht komplett transparente Brücken

Nodedarstellungen: Teiltransparente Nodes

Transparenz in normalen Nodes verursacht Probleme: Dinge dahinter werden unter Umständen nicht mehr gezeichnet (Effizienzgründe)

Darum gibt es 3 Drawtypes, die besser mit Transparenz umgehen

```
minetest.register_node("...", {
  description = "Nodebeschreibungen",
  drawtype = "glasslike|glasslike_framed|allfaces",
  tiles = {"mod_foo0.png","mod_foo1.png"},
  groups = {choppy = 2, ...}
})
```

glasslike, glasslike_framed, allfaces unterscheiden sich in der Art, wie mit Rückseiten umgegangen wird

Nodedarstellungen: Teiltransparente Nodes, Rückseiten

- glasslike: Nur die Vorderseiten werden gezeichnet. Hängen mehrere Nodes aneinander, so werden nur die Vorderseiten der zuerst zu sehenden Nodes gezeichnet Andere Nodes dahinter werden gezeichnet
- glasslike_framed: Wie glasslike, aber auch die Rückseiten werden gezeichnet; allerdings in der gleichen Ausrichtung; Randpixelzeile wird doppelt gezeichnet
- allfaces: Wie glasslike, aber Blöcke verbinden sich nicht, die Flächen werden immer gezeichnet

Nodedarstellungen: Flüssigkeiten

Mehrere Parameter für korrektes Funktionieren notwendig: Assert prüft, dass Flüssigkeiten in der Darstellung auch Flüssigkeiten im Verhalten sind

Immer zwei Flüssigkeitsnodes zusammen definieren

- Source: Quellblock, von dem die Flüssigkeit ausgeht
- Flowing: Automatische erzeugter Block für das Fließen

Definitionen beider Nodes müssen identisch sein, bis auf

- `liquidtype = "source"` $\leftrightarrow$ `liquidtype = "flowing"`
- `paramtype2 = "flowingliquid"` notwendig im Flowing-Node
- `drawtype = "liquid"` $\leftrightarrow$ `drawtype = "flowingliquid"`

In beiden muss als `liquid_alternative_source` der Name des Source-Nodes und als `liquid_alternative_flowing` der Name des Flowing-Nodes angegeben werden

Nodedarstellungen: Flüssigkeiten, weiteres

Weitere Flüssigkeitsrelevante Parameter

- `tiles` und `specialtiles` können beide für die Anzeige verwendet werden, sollten also beide definiert werden
- `liquid_renewable` sorgt dafür, dass ein neuer Source-Block entsteht, wenn unter den 5 Nachbarn entlang einer Koordinatenachse, aber nicht nach unten, zwei Source-Blöcke sind
- `alpha` Zahl zwischen 0 und 255 bestimmt Transparenz beim Blick in den Block
- `post_effect_color = {a=..., r=..., g=..., b=...}` bestimmt Farbfilter für komplette Ansicht beim Blick aus dem Block heraus
- `walkable=false, buildable_to=true, drowning=1` sind auch oft nützlich: Man kann nicht darauf gehen, man kann darin bauen, man geht darin unter

Nodedarstellungen: Plantlike, Firelike

```
minetest.register_node("...", {
  description = "Nodebeschreibungen",
  drawtype = "plantlike|firelike",
  tiles = {"mod_foo0.png"},
  groups = {choppy = 2, ...}
})
```

- Plantlike: Zwei gekreuzte Graphiken
- Firelike: Wie Plantlike, aber legt sich an Wände

Nodeverhalten

Verhalten von Nodes kann in der Nodekonfiguration auf vielfältige Weise geändert werden

- `walkable` kann man auf dem Node laufen, oder dringt der Spieler durch sie durch?
- `buildable_to` kann man hier bauen, also die Node durch eine andere direkt ersetzen?
- `drowning` so viel Schaden, wenn keine Luftblasen übrig.
- `pointable` kann der Mauszeiger darauf zeigen?
- `climbable` kann man daran hochklettern?
- `light_source` Menge an ausgesendetem Licht; `minetest.LIGHT_MAX` ist maximale Lichtstärke
- `damage_per_second` Schaden innerhalb dieses Nodes
- `floodable` kann durch Flüssigkeiten überflutet werden
- `drop` Gegenstand, der beim Abbauen erzeugt wird
- `paramtype = "light"` Node lässt Licht durch

Node hooks

An verschiedenen Stellen kann Code an ein Node gebunden werden

- `on_construct` beim Bauen
- `on_destruct` vor dem Entfernen (außer Massenoperationen, z.B. Voxeleditor)
- `after_destruct` nach dem Entfernen (außer Massenoperationen)
- `on_flood` beim Überfluten
- `can_dig` kann ein Spieler diese Node abbauen?
- `on_dig` beim Abbauen
- `on_punch` beim Schlagen der Node
- `on_rightclick` beim Rechtsklick auf die Node