

Binary Space Partitioning Trees: Konzepte und Anwendungen

Bastian Rieck

Gliederung

- 1 Motivation
- 2 BSP Trees: Der klassische Ansatz
 - Grundlagen
 - Materalgorithmus
 - Probleme und Erweiterungen
- 3 BSP Trees: Anwendungen
 - Schattenberechnung (statisch)
 - Schattenberechnung (dynamisch)
- 4 Quellen

Motivation

Fuchs et. al [3]

[...] A new algorithm for solving the hidden surface (or line) problem to more **rapidly** generate realistic images [...]

As in many applications the environment to be displayed consists of polygons many of whose relative geometrical relations are **static**, we attempt to capitalize on this by **preprocessing** the environment's database so as to **decrease** the run-time computations required to generate a scene.

Konzept

„Divide et impera“:

- 1 Wähle Ebene im Objektraum
- 2 Klassifiziere (konvexe) Polygone anhand des Halbraumes, in dem sie liegen
- 3 Setze den Prozess rekursiv in jedem Halbraum fort

Binary Space Partitioning (BSP): binäre Raumpartitionierung

↔ Binärbaum

Anwendungsgebiete

- Hidden Surface Removal: Fuchs et al. (1980)
- Beschreibung von Polyedern: Thibault & Naylor (1987)
- Spiele: Carmack et al. (1993)
- Schattenberechnung: Chrystanthou & Slater (1995), basierend auf Chin & Feiner (1989)
- Radiosity, Kollisionserkennung: Ranta-Eskola (2001)
- Raytracing: Ize et al. (2008)

Problemformulierung

Gegeben

- Komplexe 3D-Szene mit (planaren) Polygonen:
 $P = \{p_1, \dots, p_n\}$
- Betrachterposition (Blickrichtung, Blickwinkel etc.)

Ziel

Erzeuge Bild, das Szene aus Betrachterposition zeigt:

- 1 Koordinatentransformation
- 2 Wegschneiden unnötiger Polygone (Clipping)
- 3 Bilderzeugung aus übrigen Polygonen; pro Pixel:
 - 1 Polygon mit geringster Entfernung zum Betrachter suchen:
Bestimmt sichtbares Polygon am Pixel
 - 2 Farbe des Pixels bestimmen (Lichtmodelle)

Vorgehen

Idee

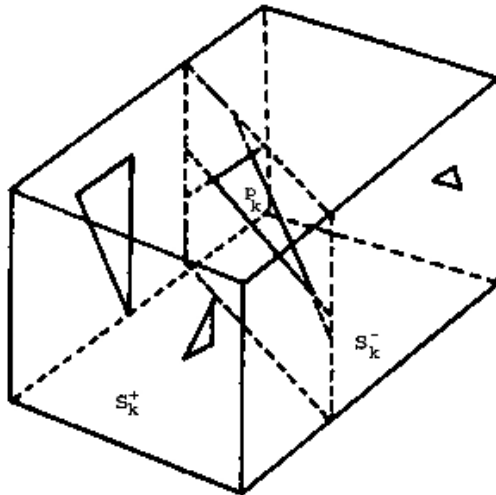
- Wähle $p_k \in P \subset \mathbb{R}^3$
- Ebene von p_k teilt Raum in zwei Halbräume $S_k^+, S_k^- \subset \mathbb{R}^3$
- Falls Betrachterposition in S_k^+ : *Kein* Polygon in S_k^- kann Polygon in S_k^+ verdecken

Algorithmus in Worten

- 1 Partitioniere $P \setminus \{p_k\}$ an der Ebene von p_k und erzeuge Halbräume. Gegebenenfalls Aufteilen (Splitting) von Polygonen.
- 2 Wiederhole Prozess mit beiden Halbräumen, bis nur noch einelementige Mengen vorliegen.

Beispiel

nach [3]





Algorithmus in Pseudocode nach [3]

```
Tree MakeTree(PolygonList pl)
{
    int k = SelectPolygon(pl);
    PolygonList PosList, NegList;
    Polygon PosParts, NegParts;
    for(int i = 0; i < pl.size (); i++)
    {
        if (i != k)
        {
            SplitPolygon(pl[i], pl[k], PosParts, NegParts);
            PosList.push_back(PosParts);
            NegList.push_back(NegParts);
        }
    }
    return(CombineTree(MakeTree(PosList),
                       pl[k],
                       MakeTree(NegList)));
}
```

Resultat

Binärbaum:

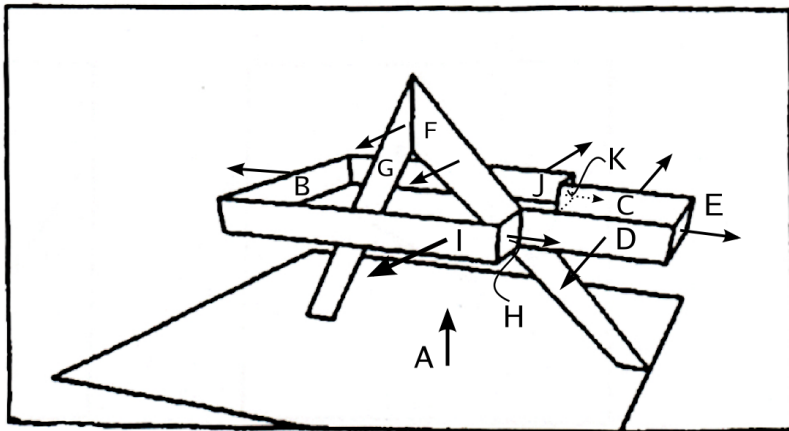
- Knoten enthält partitionierendes Polygon („Splitter“)
- Jeder Teilbaum beschreibt Halbraum

Bemerkungen:

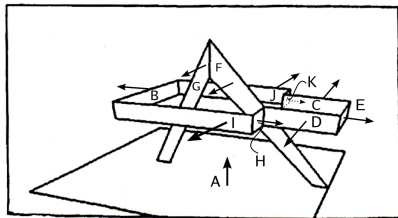
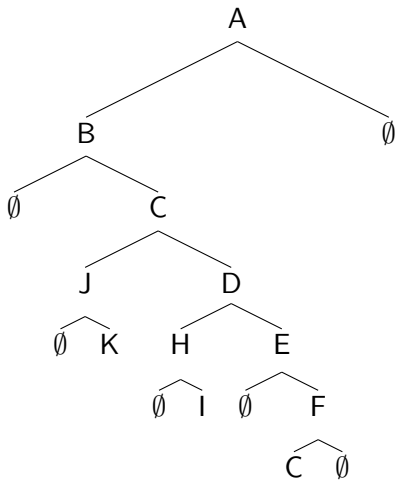
- Splitter aus Objektraum wählen: „auto-partition“
- Beliebige Splitter-Wahl möglich
- Objektdimension d beliebig. Hier nur $d = 2, 3$.

Beispiel: Objektraum

nach [3]



Beispiel: BSP Tree



Gliederung

- 1 Motivation
- 2 **BSP Trees: Der klassische Ansatz**
 - Grundlagen
 - **Maleralgorithmus**
 - Probleme und Erweiterungen
- 3 BSP Trees: Anwendungen
 - Schattenberechnung (statisch)
 - Schattenberechnung (dynamisch)
- 4 Quellen

Maleralgorithmus *ohne* BSP Trees

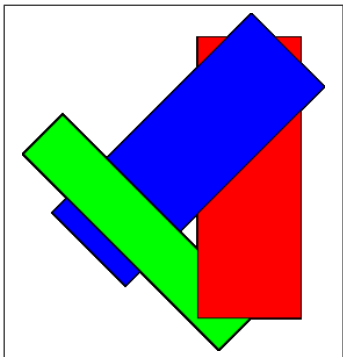
Painter's Algorithm

Ziel: Bestimme sichtbares Polygon an Pixel.

Standardalgorithmus:

- 1 Sortiere Polygone nach absteigender Distanz zur aktuellen Betrachterposition.
- 2 Sukzessives Zeichnen der sortierten Polygone.

Oh je.



Zyklische Überlappung [4]

Was nun?

- **BSP Trees** \rightsquigarrow Fuchs et al.
- Z-Buffer-Algorithmus (Catmull)

Maleralgorithmus *mit* BSP Trees

nach [3]

```

BackToFront(EyePos e, Tree t)
{
    if (CheckPos(t.root,e) == POS)
    {
        BackToFront(e,t.neg);
        Draw(t.root);
        BackToFront(e,t.pos);
    }
    else
    {
        BackToFront(e,t.pos);
        Draw(t.root);
        BackToFront(e,t.neg);
    }
}

```

- ① Habe Ebene in Wurzel
- ② Auf welcher Seite der Ebene ist der Betrachter?
- ③ Zeichne Polygone auf anderer Seite
- ④ Zeichne Wurzel
- ⑤ Zeichne Polygone auf Betrachterseite

Ergebnisse

- Schneller Algorithmus zur Sichtbarkeitsberechnung
- Splitting von Polygonen verhindert pathologische Fälle
- Berechnung der BSP Trees **vor Rendering** der Szene möglich
- BSP Tree **unabhängig** von Betrachterposition

Gliederung

- 1 Motivation
- 2 **BSP Trees: Der klassische Ansatz**
 - Grundlagen
 - Materalgorithmus
 - Probleme und Erweiterungen
- 3 BSP Trees: Anwendungen
 - Schattenberechnung (statisch)
 - Schattenberechnung (dynamisch)
- 4 Quellen

Probleme

*Zwei Seelen wohnen, ach! in meiner Brust,
die eine will sich von der andern trennen*

Johann Wolfgang von Goethe

Einerseits:

Balancierter BSP Tree:
Schnelleres Durchsuchen
möglich, aber *mehr* „Splits“

Andererseits:

Möglichst *wenig* Polygone im
BSP Tree (wenig „Splits“), dafür
unbalanciert

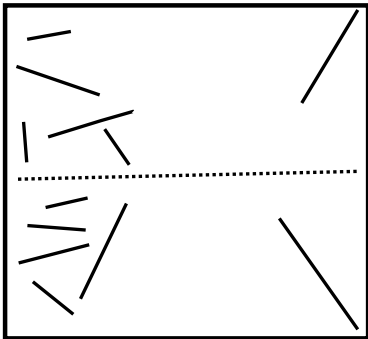
... so klug als wie zuvor

Das Problem, einen BSP Tree zu konstruieren, der *beiden* Bedingungen genügt, ist NP-vollständig.

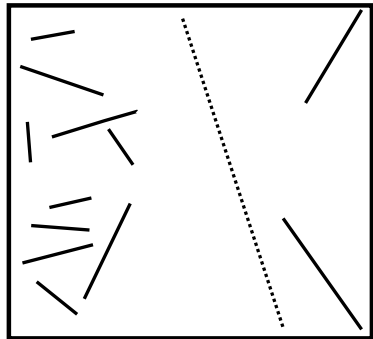
Wähle Heuristik je nach Anwendungszweck.

Das Problem mit Heuristiken

nach [5]



So?



Oder so?

BSP Trees: Erweiterungen

Dynamische Objekte

Grundidee

- *Neue* Objekte einfügen: Kein Problem
 - Objekt **löschen**: Was ist, wenn Objekt gesplittet wurde?
~> Fallunterscheidungen
 - Umsortieren nach Entfernen des Objektes (nicht trivial!)
 - Bewegliches Objekt an neuer Position in Baum einfügen
-
- Ergebnisse von Naylor (1990), Chrysanthou & Slater (1992, 1995)
 - Später mehr ~> dynamische Schattenberechnung

Gliederung

- 1 Motivation
- 2 BSP Trees: Der klassische Ansatz
 - Grundlagen
 - Materalgorithmus
 - Probleme und Erweiterungen
- 3 **BSP Trees: Anwendungen**
 - **Schattenberechnung (statisch)**
 - Schattenberechnung (dynamisch)
- 4 Quellen

Konzepte und Begriffe

Chin & Feiner [1]

Shadow Volume Schattenvolumen, das ausgehend von einer Lichtquelle in der Szene erzeugt wird

Shadow Plane Schattenebene, die von Lichtquelle und einer Kante des Polygons erzeugt wird

„In“ Im Schatten liegende Polygonfragmente

„Out“ Beleuchtete Polygonfragmente

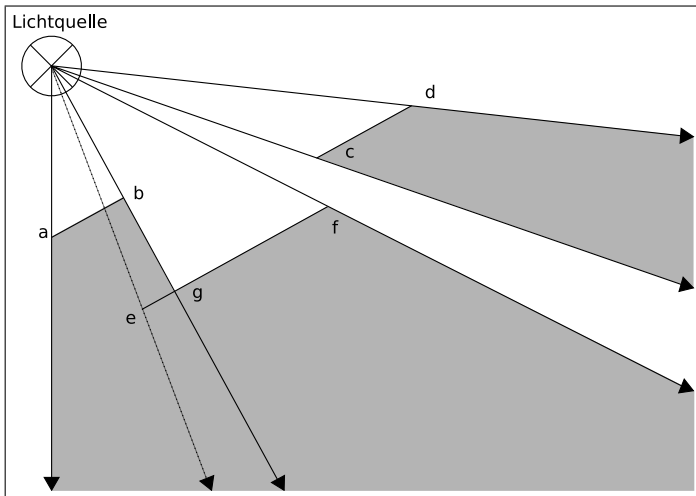
Shadow Volume BSP Trees (SVBST)

Definition

SVBSPT ist modifizierter BSP Tree:

- Innere Knoten enthalten Schattenebenen
- Blätter sind *entweder* „In“- oder „Out“-Knoten

Beispiel: Schattenvolumen nach [1]



SVBSPT Algorithmus

Voraussetzungen

- Punktlichtquelle (gerichtet oder ungerichtet)
- Statische Geometrie
- „Normaler“ BSP Tree, um Polygone in **aufsteigender** Distanz zur Lichtquelle zu sortieren

Idee

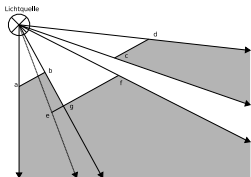
Um zu überprüfen, ob neues Polygon beleuchtet ist:

- Bilde Vereinigung aller Schattenvolumina
- **Suche** beleuchtete Teile des Polygons; **ignoriere** Teile, die bereits im Schatten liegen
- Erweitere gegebenenfalls bekanntes Schattenvolumen

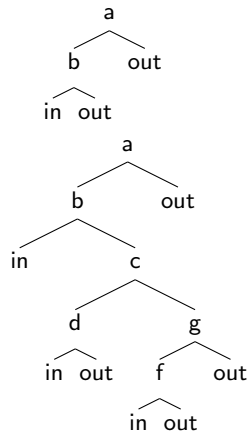
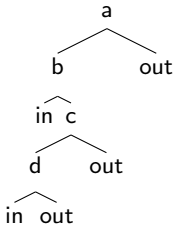
Algorithmus

- 1 Beginne mit leerem Baum („Out“) und füge iterativ Polygone hinzu
- 2 Filtere Polygon durch SVBSPT: Polygon wird durch Schattenebene partitioniert
- 3 Polygonfragment erreicht entweder „In“- oder „Out“-Knoten:
 - „In“: **Kein** Einfügen im Baum
 - „Out“: **Ersetze** Knoten durch Schattenebenen mit „Out“- und „In“-Knoten

Beispiel: 2D-SVBSPT nach [1]



out





Schattenberechnung, Pseudocode

nach [1]

```
SVBSPT ShadowGenerator(Light PLS, BSPTree B)
{
    PolygonList P = B.FrontToBack();

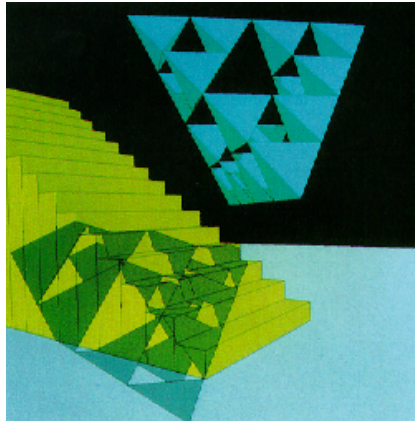
    SVBSPT S = OUT;
    for(int i = 0; i < P.size(); i++)
    {
        S = DetermineShadow(P[i], S, PLS);
    }

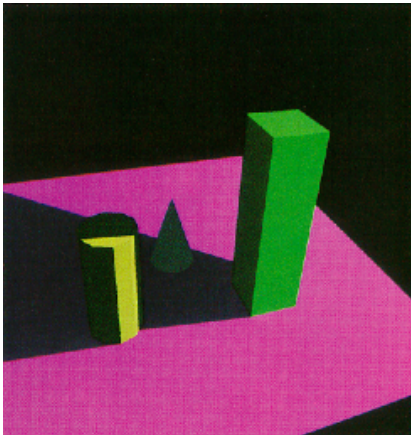
    return(S);
}
```

```
SVBSPT DetermineShadow(Polygon p, SVBSPT S, PLS)
{
    if (S == IN)
    {
        p.status = UNLIT;
        return(S);
    }
    else if (S == OUT)
    {
        p.status = LIT;
        return(MakeSVBSTNode(S, p, PLS));
    }
    else
    {
        SplitPolygon(p, S, PosPart, NegPart);
        S.NegNode = DetermineShadow(NegPart, S.NegNode, p);
        S.PosNode = DetermineShadow(PosPart, S.PosNode, p);
    }
    return(S);
}
```

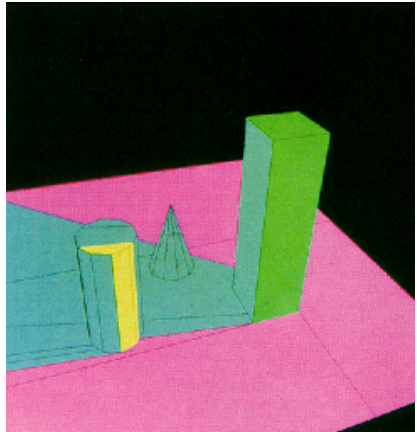
Resultate

nach [1]





Gerendertes Bild

Schattenvolumina und
Polygonfragmente

Erweiterung: Mehrere Lichtquellen

- Berechne globalen BSP Tree
- Führe Algorithmus für *jede* Lichtquelle aus (jeweils separater SVBSPT)
- Polygonfragmente merken sich, von welcher Lichtquelle sie beleuchtet werden
- Einhängen der Polygonfragmente als Kinderknoten des ursprünglichen Polygons im globalen BSP Tree

Gliederung

- 1 Motivation
- 2 BSP Trees: Der klassische Ansatz
 - Grundlagen
 - Maleralgorithmus
 - Probleme und Erweiterungen
- 3 **BSP Trees: Anwendungen**
 - Schattenberechnung (statisch)
 - **Schattenberechnung (dynamisch)**
- 4 Quellen

Dynamische Schatten

Chrysanthou & Slater [2]

- **Ungeordneter** SVBSPT
- Polygone speichern, „wo“ ihre Fragmente im Baum landen
- Zeiger („Rückwärtskanten“ im Baum), um dynamische Objekte zu löschen
- Mischen von Teilbäumen beim Löschen
- Dynamische Objekte pro Frame an neuer Position einfügen

Ungeordnete SVBSP Trees

USVBST

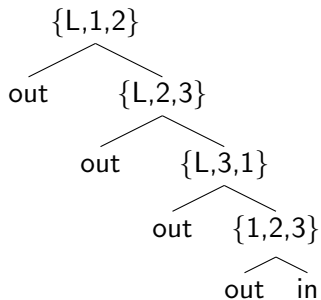
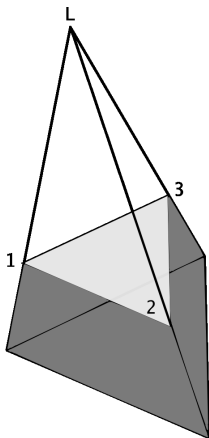
Definition

- *Keine* Sortierung der Polygone
- Partitionierung an Schattenebenen wie bei SVBSPT
- Komplettes Polygon wird mit eingefügt: „SP“-Knoten, „PP“-Knoten
- „In“-Region kann Polygone enthalten
- Knoten tiefer im Baum kann Schatten auf höheren Knoten werfen \rightsquigarrow „Target“-Beziehung

Konzeptuell wie SVBSPT. Aber: Probleme beim **Löschen** von Polygonen.

Beispiel: USVBST

nach [2]



Löschen von Polygonen und deren Schattenvolumina

Je nach Position im Baum:

„In“-Knoten Entsprechenden Knoten entfernen

Blatt Ersetze durch „Out“-Knoten; Schatten von
„Target“-Polygone entfernen

Innerer Knoten Erhalte **Teilbäume**.

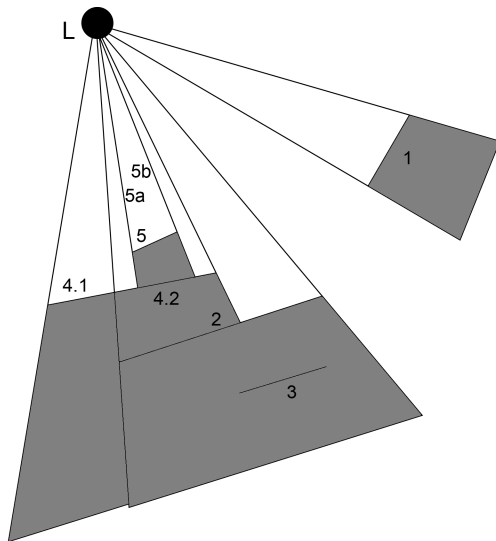
- Falls Polygon Schatten wirft: Ersetze durch Schatten von Polygonen, die in „Target“-Beziehung mit gelöschtem Polygon stehen
- Polygone im „In“-Knoten des zu löschenden Polygons neu einfügen

Vereinigung der Teilbäume

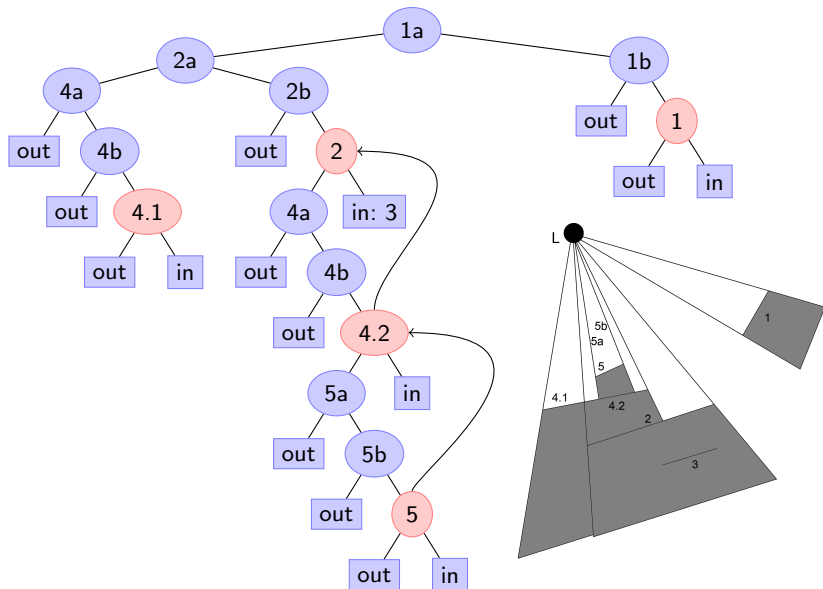
- Suche „PP“-Knoten, der zur Wurzel des *kleineren* Teilbaumes gehört
- Filtere diesen Knoten mit seinen Kinderknoten durch größeren Teilbaum
- Wenn „Out“-Knoten erreicht: Füge Schattenvolumen des „PP“-Knoten wieder ein
- Rekursiv weiter mit den Teilbäumen der „SP“-Knoten (nur positive Seite)

Beispiel

nach [2]



Passender USVBSPT



Ergebnisse

- Verfahren verbraucht deutlich **weniger** Speicher und Rechenzeit als Originalverfahren!
- Schneller als Bäume komplett neu aufzubauen
- Dynamische Transformationen in $< 0.1s$ pro Frame (1995, ≈ 60 Polygone)

Vielen Dank für die Aufmerksamkeit.

Fragen?

- [1] CHIN, NORMAN und STEVEN FEINER: *Near real-time shadow generation using BSP trees*.
In: *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, Seiten 99–106, New York, NY, USA, 1989. ACM.
- [2] CHRYSANTHOU, YIORGOS und MEL SLATER: *Shadow volume BSP trees for computation of shadows in dynamic scenes*.
In: *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, Seiten 45–50, New York, NY, USA, 1995. ACM.
- [3] FUCHS, HENRY, ZVI M. KEDEM und BRUCE F. NAYLOR: *On visible surface generation by a priori tree structures*.
In: *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, Seiten 124–133, New York, NY, USA, 1980. ACM.
- [4] MUŁA, WOJCIECH: *Painter's problem*, 2009.
[Online; Stand 4. Januar 2009].

[5] NAYLOR, BRUCE F.: *Binary Space Partitioning Trees - A Tutorial*.

In: *Proc. of SIGGRAPH '94*, 1994.