Lösung zu Aufgabe 1184 aus
Elemente der Mathematik 57 (2002), S. 133.

Michael Nüsken, Universität Paderborn.
17. Februar 2003.

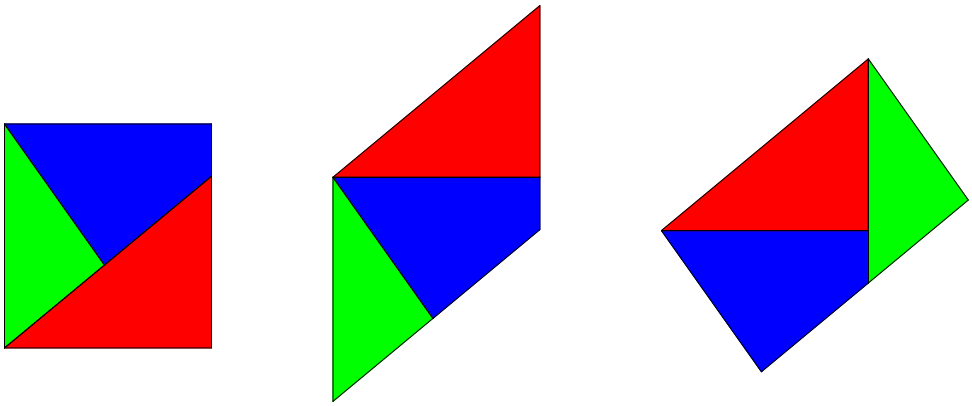
Aufgabe 1184: *Nach einem Orakelspruch sollte die Pest in Griechenlang dann zu Ende gehen, wenn der würfelförmige Altar im Apollonheiligtum auf Delos verdoppelt werde. Der Spruch verlan die (unmögliche) Konstruktion der Kante des neuen Altars mit Zirkel und Lineal. Bei Ausgrabunge auf Delos soll nun aber ein Granitquader, dessen Kanten sich wie 2:1:1 verhalten, gefunden worden sein. Damit drängt sich eine neue Deutung des Orakelspruchs auf: Ein Quader vom Typus 2:1:1 ist so (in [endlich viele] Polyeder) zu zerlegen, dass sich aus den Teilen ein Würfel bilden läs Gibt es eine solche Zerlegung?*
Wenn ja, gebe man eine

- *leicht zu realisierende Zerlegung für Steinmetze und Puzzlefreunde an.*
- *Zerlegung mit einer (möglichst) kleinen Anzahl von Stücken an.*

Ernst Specker, Zürich, CH

Meine Antwort: *Ja, es gibt eine solche Zerlegung. Eine leicht zu realisierende Lösung hat acht Teile, es genügen aber auch schon sieben.*
Die Sieben-Teile-Lösung hat eine solch schöne besondere Eigenschaft, dass ich glaube, mit wenige Teilen kann es nicht gehen.

Wie jeder, dem ich das Problem später schilderte, habe auch ich zuerst versucht, den Altar (der 2:1:1-Quader) durch achsenparallele Schnitte zu zerschneiden und die abgeschnittenen Stücke nach und nach zu einem Würfel zusammenzufügen. Sei für alles folgende a die Kantenlänge des Würfels, al die reelle dritte Wurzel aus 2. Nun wie schneiden wir: zuerst ein 1:1:1-Quader, dann drei Scheiben $1:1:(a-1)$; das ergibt schon einmal einen Würfel mit angesetzten Scheiben, sodass drei Kanten die richtige Länge haben. Es fehlen nun noch drei Stäbe $1:(a-1):(a-1)$ und ein Würfel der Kantenlänge $a-1$) und vom Altar bleibt eine etwas dickere Scheibe mit Verhältnis $1:1:(4-3a)$. Und so fort...
Als ich zu begreifen begann, dass mich das vermutlich in eine unendliche Schleife stürzen würde, habe ich überlegt, wie ich wohl beweisen könnte, dass es nicht geht. Mir ist nichts Rechtes eingefallen, selbst nicht mit Zusatzbedingungen wie, nur achsenparallele Schnitte zu erlauben, so dass ich schließlich auf die Idee gekommen bin, doch mal andere als achsenparallele Schnitte zu verwenden. Ja und dann bin ich erst einmal in die zweite Dimension zurückgegangen und habe mich an Rechtecken versucht. Und siehe da: hier war alles ganz einfach. Wenn man von einem Rechteck ein Dreieck abschneidet, das eine Ecke mit dem Rechteck gemein hat, so kann man dieses auf die andere Seite schieben und erhält ein Parallelogramm. Das rote Dreieck in der linken Figur kann man von unten nach oben schieben.



Das klappt sogar dann noch, wenn man mit einem Parallelogramm beginnt. Durch geeignete Wahl kommt man in zwei Schritten zu einem Recheck, dessen Höhe man in gewissen Grenzen wählen kann. Im obigen Bild besteht der zweite Schritt in der Verschiebung des grünen Dreiecks nach rechts oben.
Der Schritt ins Dreidimensionale ist nun auch nicht mehr schwer. Eine Rechtecksumwandlung liefert eine Umwandlung eines Quaders, wenn ich die zweidimensionale Lösung einfach auflege und den Quader wie eine Torte senkrecht zur zweidimensionalen Zeichnung zerschneide. Mit zwei solchen Schritten mit zueinander senkrechten Auflageflächen ist nun die Umwandlung eines Quaders in einen fast beliebigen anderen möglich, wenn nur das Volumen gleichbleibt. Ach ja, natürlich wären die alten Griechen mit dieser Lösung wieder nicht ganz zufrieden gewesen, denn da ja, wie wir heute wissen, prinzipiell unmöglich ist, die Länge $\sqrt{2}$ aus nur rationalen Längen zu konstruieren, kann auch unsere Lösung nicht mit Zirkel und Lineal konstruierbar sein; aber das war ja auch nicht verlangt.
Nach ein wenig Probieren haben wir nun also eine Lösung mit neun Teilen, die darauf beruht, wie man ein gegebenes Rechteck durch Zerlegen in drei Teile zu einem Rechteck vorgegebener Höhe umwandeln kann.
Mein Problem war nur, dass ich Schwierigkeiten hatte, mir vorzustellen, wie das Ganze jetzt aussieht. Ich wollte also die entstehenden Teile noch bildlich darstellen. Natürlich hätte ich das auf ein Blatt Papier zeichnen können, aber mir schwebte ein Film vor, und ich dachte, das könne doch nicht so schwer sein. Mehr dazu gleich.

Es zeigte sich, dass *eines der neun Teile leer* ist. Zunächst sah es außerdem so aus, als würden zwei der verbliebenen acht Teile am Anfang und am Ende gleich zueinander liegen und sie daher zu einem Teil verschmolzen werden könnten. Eine genauere Untersuchung zeigte aber, dass das nicht ganz stimmt: die beiden Teile gleiten um eine Winzigkeit aneinander vorbei, die gerade ihrem Größenunterschied entspricht. Irgendwie ist diese winzige Verschiebung ungerecht, ich wollte noch einen Versuch machen, es doch hinzukriegen. Ein weiteres Teil kommt dabei ins Spiel, welches einmal diesseits und einmal jenseits des leicht rutschenden Teils liegt und genau den Rutschbetrag ausgleicht. Schneiden wir doch ein Scheibchen vom einen ab und kleben es an das andere dran! Dann sollte die Verschiebung nicht mehr nötig und die Verschmelzung möglich sein. Dieser Plan funktioniert! *Es genügen also sieben Teile*Der Film ganz am Ende zeigt das Ergebnis.

Nun kann ich endlich folgende überraschende und wunderschöne Beobachtung festhalten:
Betrachtet man die Flächen der sieben Teile, so ist jede entweder im Altar oder im Würfel Teil der Aussenfläche.
Keine Fläche ist beide male im Innern verborgen oder beide Male aussen.
Unter der Bedingung, dass die Teile nur parallelverschoben und nicht gedreht werden, und Würfel

und Altar keine parallelen Kanten haben ---wie das in der gefundenen Lösung der Fall ist---, ist es auch gar nicht möglich, dass eine Fläche zweimal aussen ist.
Ich könnte mir vorstellen, dass diese Beobachtung belegt, dass es nicht mit weniger als sieben Teilen gehen kann. Aber ich habe weder einen formalen Beweis noch eine Idee, wie ein solcher aussehen könnte.

Im folgenden gehe ich auf einige Details genauer ein und am Ende finden sich noch einige andere Illustrationen.

Die Details

Wir (re)initialisieren die Arbeitsmaschine und laden die Zeichenpakete.

```
> restart; with(plots); with(plottools); with(LinearAlgebra)
Warning, the name changecoords has been redefined

Warning, the name arrow has been redefined
```

Das analoge zweidimensionales Problem

Zuerst betrachten wir, wie ein gegebenes Rechteck zerschnitten und zu einem Rechteck gegebener Höhe zusammengesetzt werden kann.

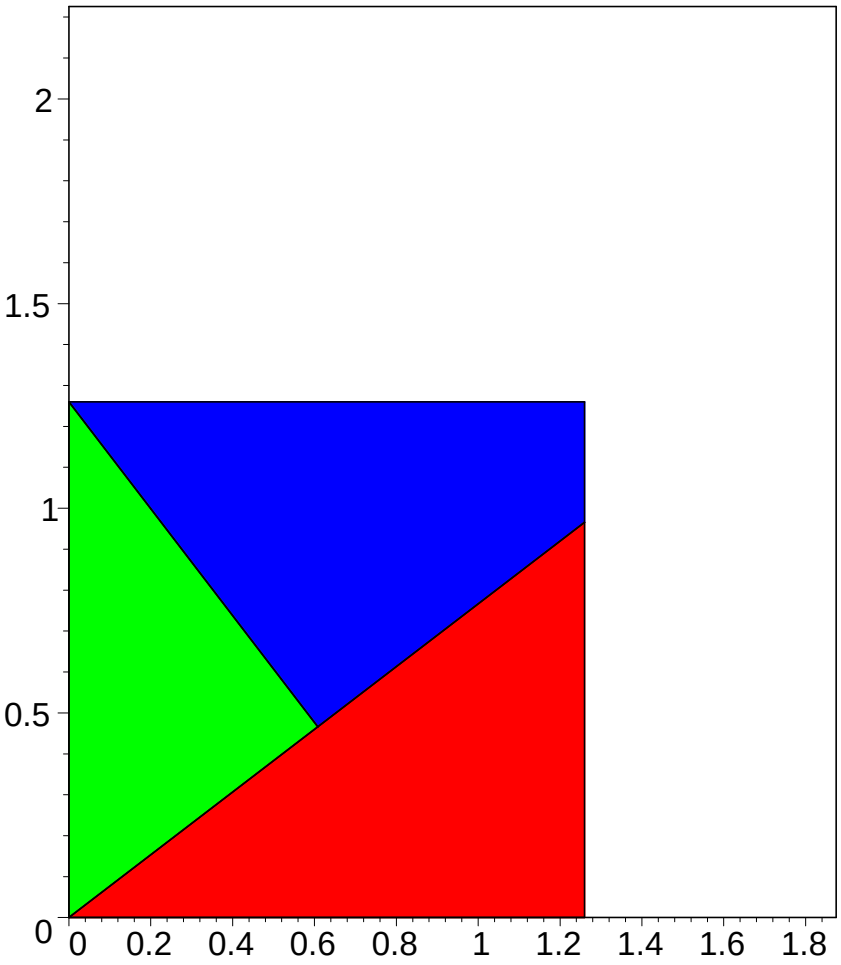
Diese Prozeduren erzeugen kurze Filme, die das demonstrieren.

```
> rectangle_data := proc(a, b, h)
      local m, c, d, e;
          m := sqrt((b / h)^2 – 1);
          c := m*h^2 / b;
          d := m*c;
          e := a*d / c;
          c, d, e, arctan(e / a)
      end proc;
  rectangle_parts := proc(a, b, c, d, e)
          polygon([[0, 0], [a, 0], [a, e]], color = red),
          polygon([[0, 0], [c, d], [0, b]], color = green),
          polygon([[c, d], [a, e], [a, b], [0, b]], color = blue)
      end proc;
  transform_rectangle := proc(a, b, h, s)
      local m, c, d, e, R, G, B, aim, α;
          c, d, e, α := rectangle_data(a, b, h);
          R, G, B := rectangle_parts(a, b, c, d, e);
          aim := display([translate(R, 0, b), translate(G, a, e), B]);
          seq(display([translate(R, 0, b*i / s), G, B]), i = 0 .. s),
              seq(display([translate(R, 0, b), translate(G, a*i / s, e*i / s), B]), i = 1 .. s),
              seq(rotate(aim, –α*i / s, [0, b]), i = 1 .. s);
          display(%, scaling = CONSTRAINED, axes = BOXED, insequence = true)
      end proc
```

Dieser erste Film zeigt wie die Umwandlung eines Quadrats mit Seitenlänge $2^{\left(\frac{1}{3}\right)}$ in ein Rechteck der Höhe 1 erfolgt: Wir schneiden zunächst das rote Dreieck ab und schieben es die obere Seite. Vom entstandenen Parallelogramm schneiden wir das grüne Dreieck ab und schieben es an die rechte obere Seite. Da das grüne Dreieck rechtwinklig gewählt

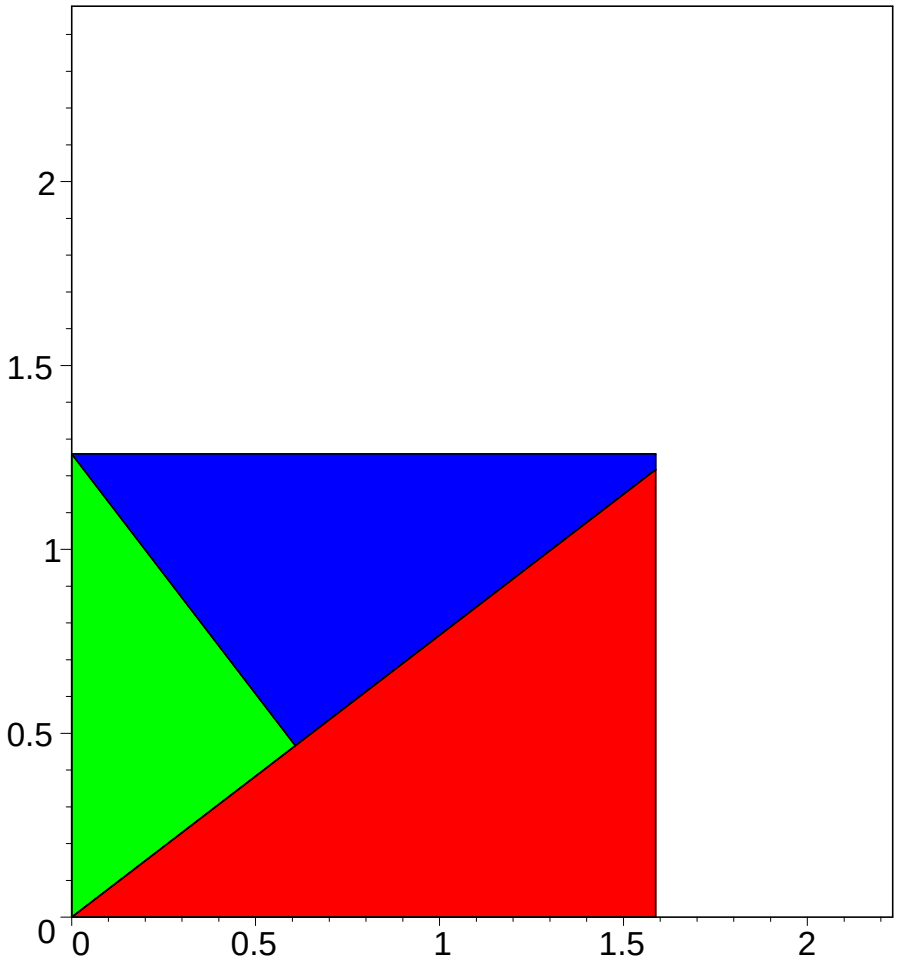
wurde, haben wir jetzt wieder ein Rechteck. Die Höhe konnten wir uns bei dieser Konstruktion innerhalb gewisser Grenzen wünschen: sie sollte hier 1 sein.

```
> transform_rectangle(2^(1/3), 2^(1/3), 1, 5)
```



Der nächste Film zeigt genau so eine Umwandlung, nur dass wir diesmal mit einem $2^{\left(\frac{2}{3}\right)}$ x $2^{\left(\frac{1}{3}\right)}$-Rechteck starten.

```
> transform_rectangle(4^(1/3), 2^(1/3), 1, 5)
```



>

Erste 3D-Zeichenversuche

Nach diesen ersten Zeichenversuchen habe ich schließlich entschieden, dass ich ein Werkzeug brauche, um konvexe Hüllen zu zeichnen.

Konvexe Hülle mit Gewalt

Farben suchen und festlegen

Programmierung der Acht-Teile-Lösung ...

Nun wir wollen einen Film erzeugen, der zeigt wie der 2:1:1-Quader auseinandergenommen und zu einem Würfel zusammengesetzt wird.
Die Parameter f, mv und S kontrollieren die Bewegungen im Film. Die Flexibilität ist hilfreich, um später unterschiedliche Abläufe mit ein und demselben Programm zu erzeuge
f(n,t) definiert wie die Zeit für das Teil n mit einer globalen Zeit t fortschreitet. (Wir könne zum Beispiel ein Teil von t=0 bis t=1/2 festlassen und dann mit doppelter Geschwindigkeit bewegen oder ähnliches.)

mv(t,A,B1,B2,B3,B4,B5,C,DA,DB,DC) definiert die tatsächliche Bewegung: A ist die Startposition, C die Zielposition und B3 die Zwischenposition. Die anderen Bs können helfen, Durchdringungen zu vermeiden. Die Ds sind für Sprengzeichnungen gedacht. S+1 ist die Anzahl gewünschter Bilder. Die globale Zeit läuft von 0 bis 1 in Schritten von 1/S während des Films.

Die Prozedur erzeugt zunächst 3D-Zeichnungen von jedem Teil. Dazu haben wir auf dem Papier die jeweiligen Ecken bestimmt und das obige Programm zur Berechnung und zum Zeichnen konvexer Hüllen übernimmt den Rest. Die Variablen RR, RG, RB, GR, GG, GB BR, BG, BB enthalten die Zeichnungen, und zwar enthält XY den Schnitt zweier Prismen, wobei sich X und Y auf die Farbe (rot,grün,blau) eines Teils der zweidimensionalen Lösungen von oben beziehen. Das Teil GG ist leer. Für jedes Bild des Films werden diese Teile werden verschoben und zu einem Bild zusammengesetzt. Dabei werden die Prozeduren mv und f verwendet, um die Verschiebungsvektoren zu berechnen.

Achtung: Alle Koordinatenangaben beziehen sich auf die mittleren Situation des a^2:a:1 -Quaders gemacht, da nur hier beide Schnittrichtungen richtig erkennbar sind.

```
> cut_cube := proc(f, mv, S)
  local a, c, d, e, α, s, c2, d2, e2, A, y, B, z, C, RR, RG, RB, GR, GG, GB, BR, BG, BB, RX,
  GX, BX, XR, XG, XB, RX1, GX1, BX1, XR1, XG1, XB1, RX2, GX2, BX2, XR2, XG2, XB2,
  X0, Y0, Z0, X1, Y1, Z1, L, t;
      a := 2^(1 / 3);
      c, d, e, α := rectangle_data(a^2, a, 1);
      s := sqrt(a^2 − 1);
      e2 := expand(a^2 − s);
      c2, d2 := op(expand((1 − s)*[a^2, 1] + s*[e2, 0]));
      A := [e2, e*e2 / (a^2), 0];
      expand(λ*[a^2, e, 1] + (1 − λ)*A − [c2, y, d2]);
      convert(%, set);
      solve(%);
      y := simplify(subs(%, y));
      B := [c2, y, d2];
      expand(λ*[0, 0, 1] + (1 − λ)*B − [c, d, z]);
      convert(%, set);
      solve(%);
      z := simplify(subs(%, z));
      C := [c, d, z];
      RR := polyhedron("RR", [[0, 0, 1], [a^2, 0, 1], [a^2, e, 1], [c2, 0, d2], B],
          color = RRcolor);
      userinfo(4, delos, "RR", print(%));
      RG := polyhedron("RG", [[0, 0, 1], [c, d, 1], [0, a, 1], C], color = RGcolor);
      userinfo(4, delos, "RG", print(%));
      RB := polyhedron("RB",
          [[0, a, 1], [c, d, 1], [a^2, e, 1], [a^2, a, 1], C, B, [c2, a, d2]], color = RBcolor);
      userinfo(4, delos, "RB", print(%));
      GR := polyhedron("GR",
          [[e2, 0, 0], [a^2, 0, 0], [a^2, e, 0], A, [a^2, 0, 1], [a^2, e, 1]], color = GRcolor);
      userinfo(4, delos, "GR", print(%));
      GG := NULL;
```

```
GB := polyhedron("GB",
      [A, [a^2, e, 0], [a^2, a, 0], [e2, a, 0], [a^2, e, 1], [a^2, a, 1]], color = GBcolor);
userinfo(4, delos, "GB", print(%));
BR := polyhedron("BR", [[0, 0, 0], [0, 0, 1], [c2, 0, d2], [e2, 0, 0], A, B],
      color = BRcolor);
userinfo(4, delos, "BR", print(%));
BG := polyhedron("BG", [[0, 0, 0], [0, 0, 1], C, [c, d, 0], [0, a, 0], [0, a, 1]],
      color = BGcolor);
userinfo(4, delos, "BG", print(%));
BB := polyhedron("BB", [[0, a, 0], [0, a, 1], C, [c, d, 0], A, [e2, a, 0], [c2, a, d2], B],
      color = BBcolor);
userinfo(4, delos, "BB", print(%));
XR := [0, a, 0];
XR1 := [a^2, a, 0];
XR2 := [a^2, e, 0];
XG := [a^2, e, 0];
XG1 := [a^2, e, 0] – [0, a, 0];
XG2 := [c, d, 0] – [0, a, 0];
XB := [0, 0, 0];
XB1 := XB;
XB2 := XB;
RX := [e2, 0, 0] – [a^2, 0, 1];
RX1 := [e2, 0, 0] – [0, 0, 1];
RX2 := [c2, 0, d2] – [0, 0, 1];
GX := –[a^2, 0, 0];
GX1 := –[a^2, 0, 1];
GX2 := [e2, 0, 0] – [a^2, 0, 1];
BX := [0, 0, 0];
BX1 := BX;
BX2 := BX;
X0 := A – [c, d, 0];
X0 := X0 / LinearAlgebra:-Norm(convert(X0, Vector), 2);
Y0 := [0, a, 0] – [c, d, 0];
Y0 := Y0 / LinearAlgebra:-Norm(convert(Y0, Vector), 2);
Z0 := C – [c, d, 0];
Z0 := Z0 / LinearAlgebra:-Norm(convert(Z0, Vector), 2);
X1 := [0, a, 1] – [c2, a, d2];
X1 := X1 / LinearAlgebra:-Norm(convert(X1, Vector), 2);
Y1 := [e2, a, 0] – [c2, a, d2];
Y1 := Y1 / LinearAlgebra:-Norm(convert(Y1, Vector), 2);
Z1 := B – [c2, a, d2];
Z1 := Z1 / LinearAlgebra:-Norm(convert(Z1, Vector), 2);
L := NULL;
for t from 0 by 1 / S to 1 do L := L, display([translate(RR, mv(f(1, t), XR, XR1, XR2,
      [0, 0, 0], RX2, RX1, RX, Y0 + Z0, [0, –1, 1], –X1 + Y1 + Z1)), translate(RG, mv(
      f(2, t), XG, XG1, XG2, [0, 0, 0], RX2, RX1, RX, X0 – Y0 + Z0, [–1, 0, 1], Y1)),
```

```
        translate(RB, mv(f(3, t), XB, XB1, XB2, [0, 0, 0], RX2, RX1, RX, –X0 – Y0 + Z0,
        [0, 1, 1], –X1 + Y1 – Z1)), translate(GR, mv(f(4, t), XR, XR1, XR2, [0, 0, 0],
        GX2, GX1, GX, X0 + Y0 – Z0, [1, –1, –1], X1 + Z1)), translate(GB, mv(f(5, t),
        XB, XB1, XB2, [0, 0, 0], GX2, GX1, GX, –Y0 – Z0, [1, 1, –1], X1 – Z1)),
        translate(BR, mv(f(6, t), XR, XR1, XR2, [0, 0, 0], BX2, BX1, BX, –X0 + Y0 – Z0,
        [–1, –1, –1], –X1 – Y1 + Z1)), translate(BG, mv(f(7, t), XG, XG1, XG2, [0, 0, 0],
        BX2, BX1, BX, X0 – Y0 – Z0, [–1, 0, –1], –Y1)), translate(BB, mv(f(8, t), XB,
        XB1, XB2, [0, 0, 0], BX2, BX1, BX, –X0 – Y0 – Z0, [0, 1, –1], –X1 – Y1 – Z1)),
        NULL])
    end do;
    display([L], insequence = true, scaling = CONSTRAINED, orientation = [–60, 45],
        args[4 .. –1])
end proc
```

Verschiedene mögliche Bewegungsarten

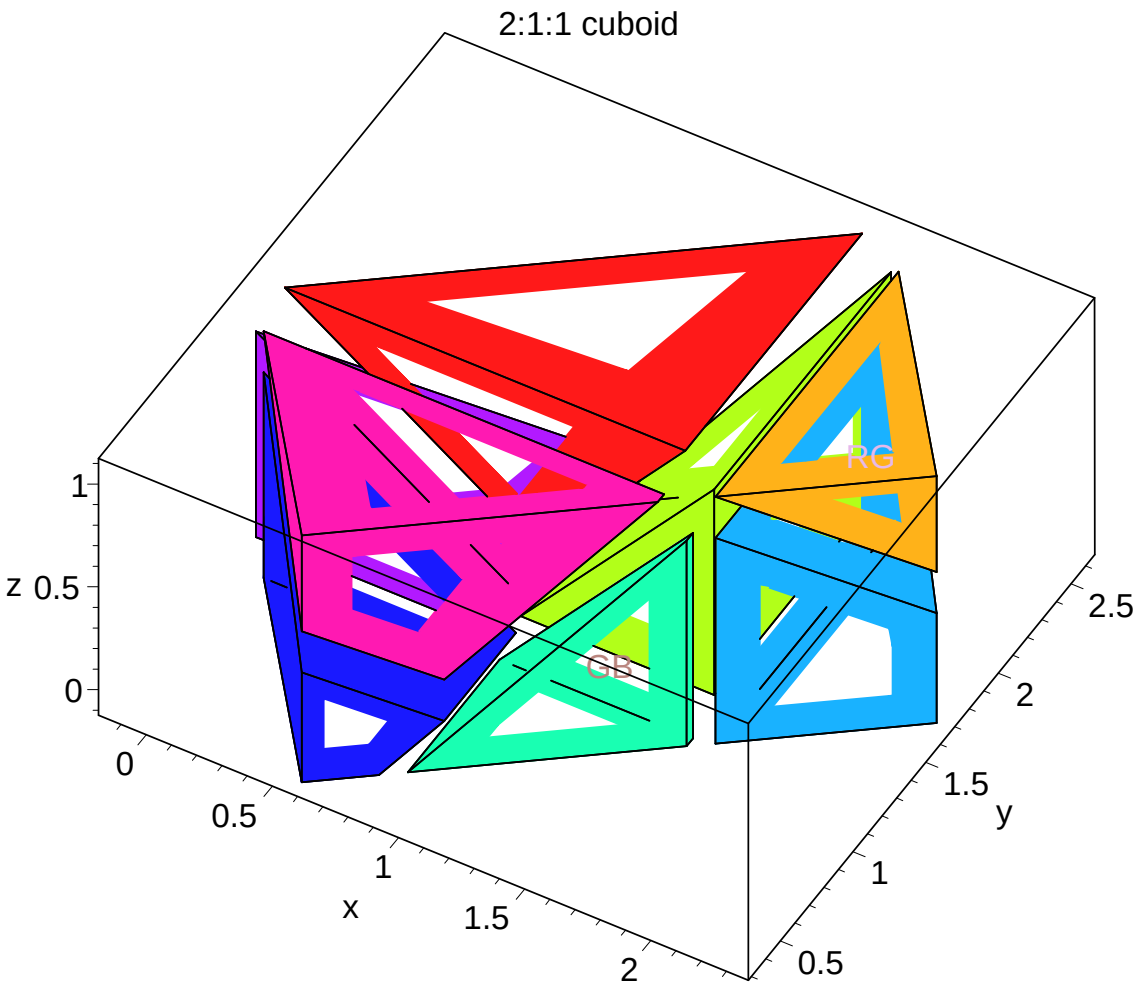
Verschiedene mögliche lokale Zeitabläufe

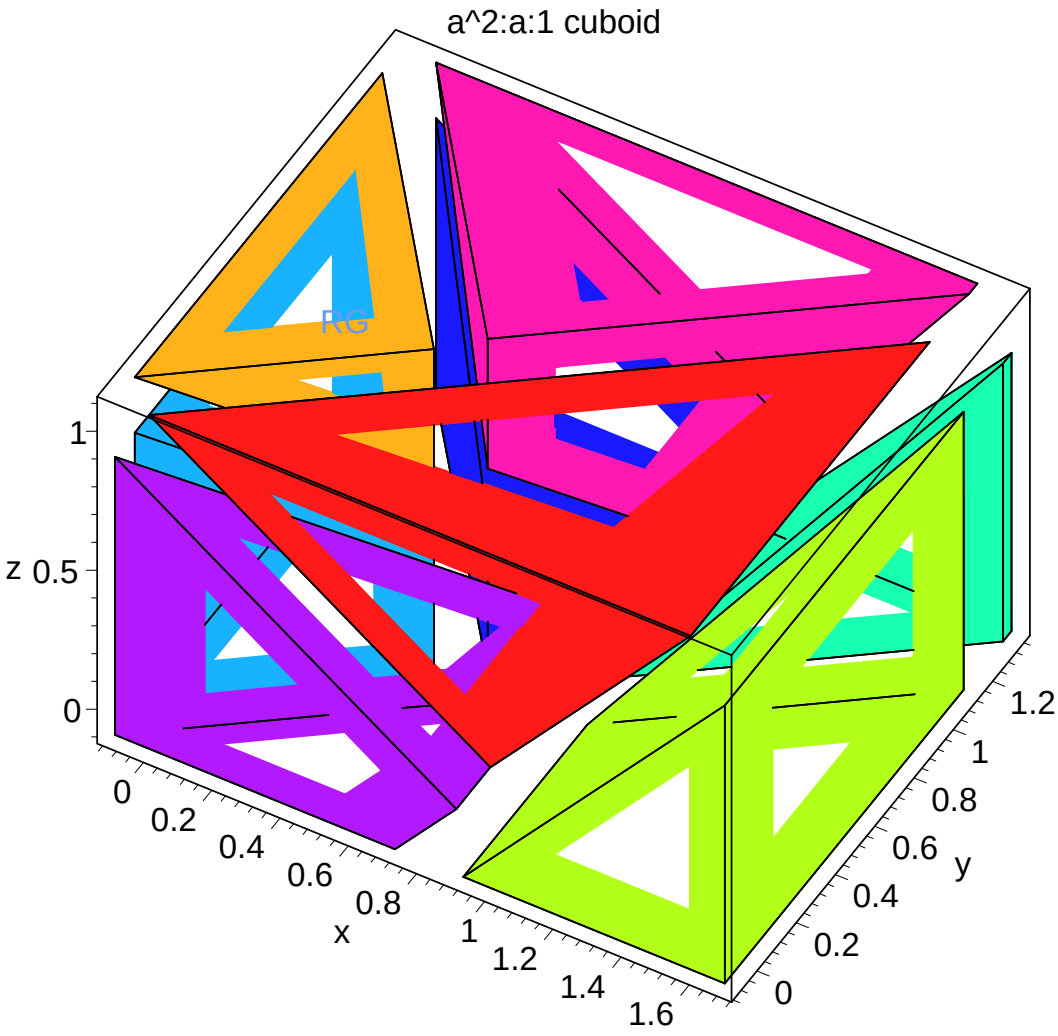
Hiermit wird ein Einzelbild erzeugt, in dem alle Flächen aufgeschnitten sind.

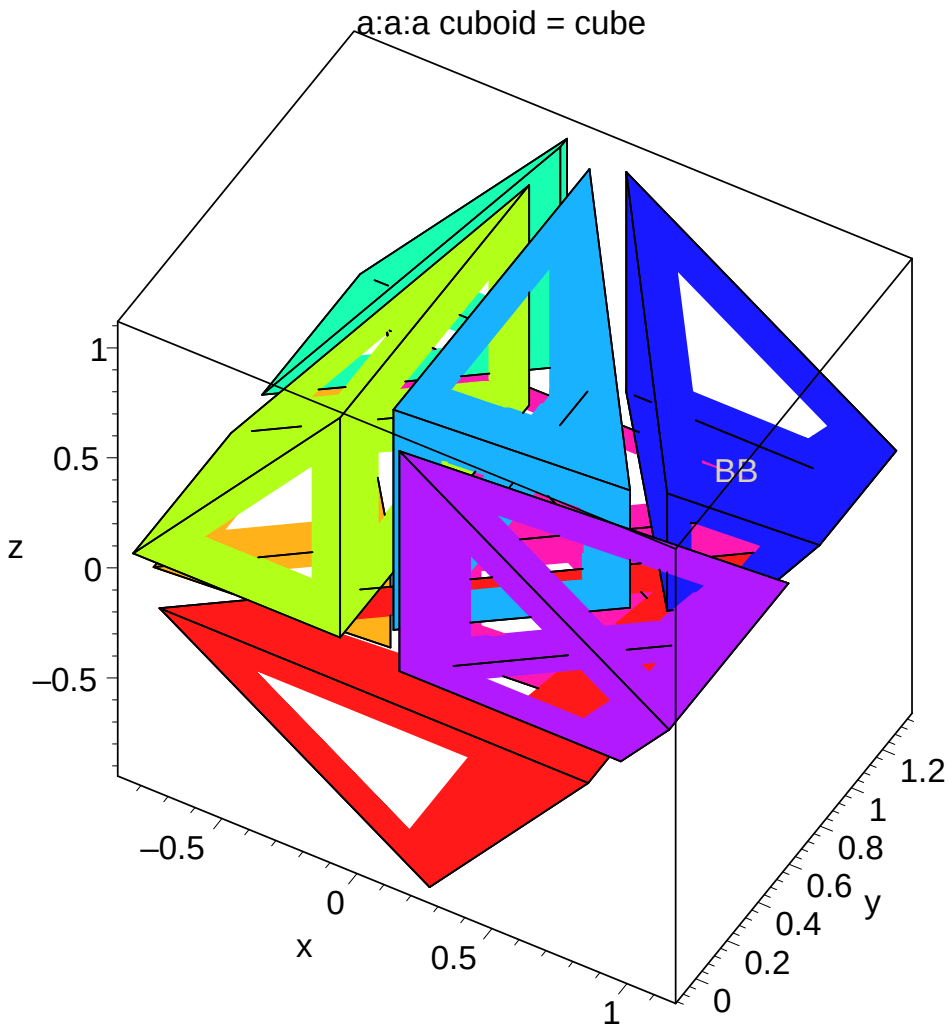
```
> show_open := proc(T)
  local f, mv;
      f := proc(n, t) T end proc;
      mv := proc(t, A, B1, B2, B3, B4, B5, C, DA, DB, DC)
              if t < 0 then A
              elif t < 1 then mv0l(t, A, B1, B2, B3, B4, B5, C)
              elif t < 2 then A + (t – 1)*DA
              elif t < 3 then B3 + (t – 2)*DB
              elif t < 4 then C + (t – 3)*DC
              else C
              end if;
              op(evalf(%))
          end proc;
      cut_cube(f, mv, .5)
  end proc
```

Standbilder leicht gesprengter Quader. (Start-, Mittel- und Zielposition)

```
> for i to 3 do
      show_open(i + .1);
      mycutout(%, .6);
      print(op(0, %)(op(%), AXESLABELS("x", "y", "z"), AXES(BOXED),
          ORIENTATION(–60, 45),
          TITLE(["2:1:1 cuboid", "a^2:a:1 cuboid", "a:a:a cuboid = cube"][i])))
  end do
```


2:1:1 cuboid
RG
GB
z
1
0.5
0
0
0.5
1
1.5
2
x
0.5
1
1.5
2
2.5
y


a^2:a:1 cuboid
RG
z
1
0.5
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
x
0
0.2
0.4
0.6
0.8
1
1.2
y



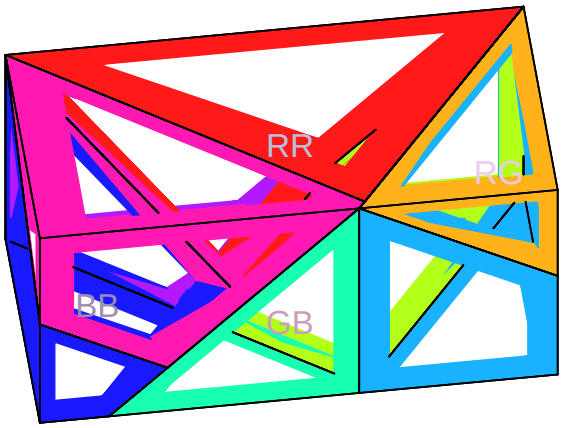
Zusammenfassung der vorliegenden Zeit- und Bewegungsfunktionen.

```
> select(proc(x)
             convert(x, string);
              not "_" in convert(%, list) and %[1 .. 2] = "mv" or length(x) < 3 and %[1 .. 1] = "f";
             evalb(%)
         end proc, [anames( )]);
   sort(%);
   map(proc(x)
             printf("%-4s: %s
             " , x, op(5, op(x)), "??")
         end proc, %)
f0  : Move all at the same time.
f1  : Move parts grouped logically.
f2  : Move one after the other.
f3  : Move first half one after the other, then second half one after the other
.
mv0c: Movement in two cubic steps passing through all intermediate points. The
passing times depend on the distance of successive points.
```

```
mv0l: Movement in six linear steps.
mv1 : Direct linear movement.
mv1b: Two linear movements.
mv2 : Quadratic movement via the middle position.
mv2b: Two quadratic movements.
mv4 : Biquadratic movement.
```

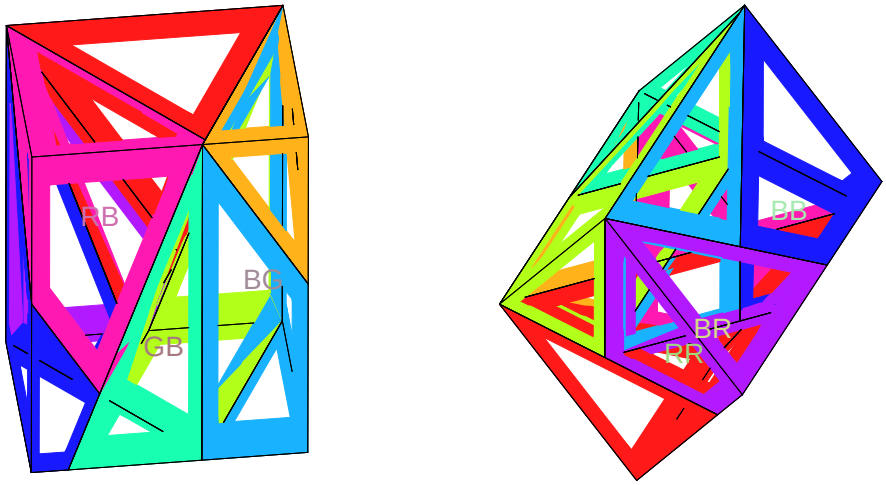
Ein Film mit zwei Bilder: Start und Ziel.

> cut_cube(*f0*, *mv1*, 1); mycutout(*%*, .7)



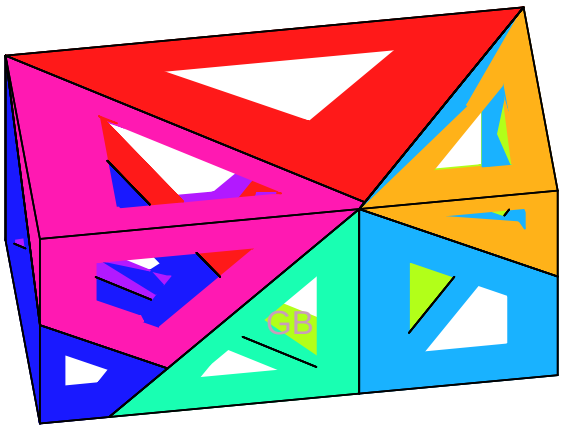
Für den Ausdruck die Filmbilder nochmal nebeneinander.

> display(*%*, *insequence* = *false*, *scaling* = *unconstrained*)



Nun ein Film, in dem sich alle Teile gleichzeitig auf geradlinigem Weg von ihrer Start- zur Zielposition bewegen.
Beachte, dass dabei das stahlblaue (GR) und das grasgrüne (BG) Teil scheinbar immer beieinander bleiben! Wir können sie also vielleicht zu *einem* Teil verschweißen, und brauchen dann nur sieben Teile. (Den Zwischenzustand können wir dann allerdings nicht herstellen.)
Ausserdem können wir hier eine sehr schöne Eigenschaft der Siebenteilezerlegung erkennen: Alle Facetten, die am Anfang im Innern lagen, liegen am Ende außen und umgekehrt. Das ist schön!

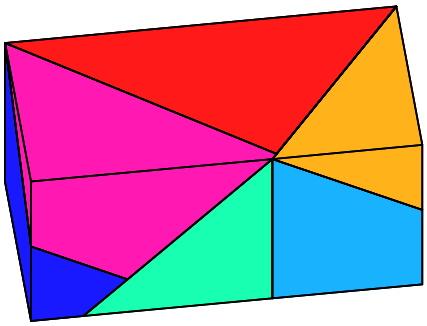
```
> cut_cube(f0@looptime2, mv1, 40); mycutout(%, .5)
```




```
> plotsetup(window):
  cut_cube(f0@looptime2,mv1,40,title="8 Teile Lösung, Geradlinige
  Bewegung von Altar zu Würfel"):
  eval( %, TEXT=(()->NULL) ):
  mycutout(%,.5);
  plotsetup(default):
```

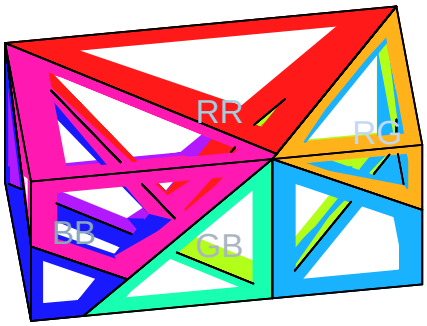
Dieser Film zeigt, wie ich die Lösung entwickelt habe.

```
> cut_cube(f1, mv0l, 96)
```



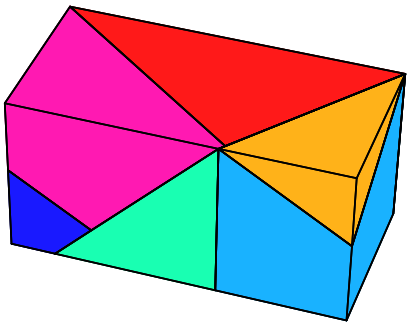
Nochmal mit offenen Flächen.

```
> mycutout(%, .7)
```



Und jetzt nochmal dieselbe Bewegung mit runderen Bewegungen.

> cut_cube(*f1@looptime2*, *mv0c*, 196, *orientation* = [–33, 50], *projection* = *normal*)



Ausgabe in getrenntes Fenster zum Abspeichern in "EdM-7Teile-8TeileFilm.mws".

```
> plotsetup(window):
  cut_cube(f1@looptime2,mv0c,196,
    orientation = [-33, 50],projection = normal,title="8 Teile
  Lösung"):
  eval( %, TEXT=(()->NULL) );
  plotsetup(default):
> # vrml( %, "Edm-7Teile-8TeileFilm.vrml" );# kann VRML keine
  Filme??
```

Programmierung der Sieben-Teile-Lösung ...

Nebenrechnung 1

Bestimmen einen geeigneten zweiten Punkt für das Teil RG.

```
> # Oberseite des Teils BG in Startposition
  E1:=[[a^2,e,1],[a^2,a+e,1],[c+a^2,d+e,z]];
```

```
# Oberkante des Teils BB verbunden mit dem zweiten
Zwischenpunkt von RG
E2:=[[c,d,z],[0,a,1],[e2-a^2+c,d,0]];
# Ebene durch Start-, zweite Zwischen- und Zielposition
der unteren Ecke von RG
E3:=[[a^2+c,e+d,z],[e2-a^2+c,d,0],[e2-a^2+c,d,z-1]];#[a^2+
c,e+d,1]];
C:=[
x-add(a1||i*E1[i],i=1..3), [1-add(a1||i,i=1..3)],
x-add(a2||i*E2[i],i=1..3), [1-add(a2||i,i=1..3)],
x-add(a3||i*E3[i],i=1..3), [1-add(a3||i,i=1..3)]
]:
C:=map(op@expand,subs(x=[x1,x2,x3],C));
#C:=subs(zip((x,y)->x=y,[c,d,e,alpha],[rectangle_data(a^2,
a,1)]),C):
S:=solve(convert(C,set),{a11,a12,a13,a21,a22,a23,a31,a32,a
33,x1,x2,x3}):;
xS:=simplify((subs(S,[x1,x2,x3])));
```

$$E1 := [[a^2, e, 1], [a^2, a+e, 1], [c+a^2, d+e, z]]$$

$$E2 := [[c, d, z], [0, a, 1], [e2-a^2+c, d, 0]]$$

$$E3 := [[c+a^2, d+e, z], [e2-a^2+c, d, 0], [e2-a^2+c, d, z-1]]$$

$$C := [-c\,a13 - a^2\,a13 - a^2\,a12 - a^2\,a11 + x1,$$
$$-d\,a13 - e\,a13 - a\,a12 - e\,a12 - e\,a11 + x2, -z\,a13 - a12 - a11 + x3,$$
$$1 - a11 - a12 - a13, -e2\,a23 + a^2\,a23 - c\,a23 - c\,a21 + x1,$$
$$-d\,a23 - a\,a22 - d\,a21 + x2, -a22 - z\,a21 + x3, 1 - a21 - a22 - a23,$$
$$-e2\,a33 + a^2\,a33 - c\,a33 - e2\,a32 + a^2\,a32 - c\,a32 - c\,a31 - a^2\,a31 + x1,$$
$$-d\,a33 - d\,a32 - d\,a31 - e\,a31 + x2, -z\,a33 + a33 - z\,a31 + x3, 1 - a31 - a32 - a33]$$

$$xS := [(3\,z\,c\,d\,e2\,a^2 + 2\,e2\,z\,c\,a^2\,e - 2\,a^7 - 2\,a^6\,z\,d - a^2\,z\,d\,e2^2 - z\,c\,d\,e2^2 - 2\,z\,c\,d\,a^4$$
$$- 3\,z\,c\,a^3\,e2 + z\,c\,a\,e2^2 + 3\,a^4\,z\,d\,e2 - e2^2\,z\,c\,e - 2\,z\,c^2\,e2\,e + 2\,z\,c^2\,a^2\,e - 2\,e2\,c\,a^2\,e$$
$$+ 2\,z\,c\,a^5 + e2^2\,z\,a^3 - 3\,e2\,z\,a^5 - a^4\,z\,c\,e - 3\,d\,a^4\,e2 + d\,a^2\,e2^2 - a^3\,e2^2 + 3\,a^5\,e2$$
$$+ 2\,a^7\,z + 2\,d\,a^6 + c\,e2^2\,e + a^4\,c\,e + 2\,c^2\,a^2\,z\,d + c^2\,e2\,z\,a - c^2\,e2\,z\,d - 3\,c\,e2\,a^2\,d$$
$$- 2\,c\,a^5 + 3\,c\,e2\,a^3 + 2\,c\,a^4\,d + c\,e2^2\,d - c\,e2^2\,a + c^2\,e2\,e - c^2\,a^2\,e - 2\,c^2\,a^3\,z - z\,c^3\,e)$$
$$/ (-2\,c\,a^3\,z - z\,c^2\,e - 2\,a^4\,z\,d + c\,e2\,e - e2^2\,z\,d + e2^2\,z\,a - c\,a^2\,e - 3\,e2\,a^2\,d + 2\,a^4\,d$$
$$+ e2^2\,d - e2^2\,a + 3\,e2\,a^3 - 3\,a^3\,z\,e2 + 2\,a^5\,z - 2\,a^5 + 3\,a^2\,z\,e2\,d - c\,e2\,z\,e + c\,a^2\,z\,e$$
$$+ 2\,c\,a^2\,z\,d + c\,e2\,z\,a - c\,e2\,z\,d), (e\,c\,e2\,z\,a + 2\,e\,a^5\,z + d\,c\,e2\,z\,a + 2\,d\,c\,a^2\,z\,e$$
$$+ 3\,d\,a^2\,z\,e2\,e - 2\,d\,c\,e2\,z\,e + d\,c\,e2\,e - d\,c\,a^2\,e - 2\,d\,a^4\,z\,e - d\,e2^2\,z\,e + d\,e2^2\,z\,a$$
$$- 3\,d\,e2\,a^2\,e - 2\,d\,c\,a^3\,z - d\,z\,c^2\,e - 3\,d\,a^3\,z\,e2 + 3\,a^2\,z\,e2\,d^2 + 2\,c\,a^2\,z\,d^2 - c\,e2\,z\,d^2$$
$$+ e\,e2^2\,z\,a - 3\,e\,a^3\,z\,e2 - e\,c\,a^3\,z + 2\,a^4\,d^2 + e2^2\,d^2 - 2\,d\,a^5 - 2\,e\,a^5 + 3\,e\,e2\,a^3$$
$$- e\,e2^2\,a + 3\,d\,e2\,a^3 - d\,e2^2\,a + 2\,d\,a^5\,z + d\,e2^2\,e + 2\,d\,a^4\,e - 3\,e2\,a^2\,d^2 - e2^2\,z\,d^2$$
$$- 2\,a^4\,z\,d^2) / (-2\,c\,a^3\,z - z\,c^2\,e - 2\,a^4\,z\,d + c\,e2\,e - e2^2\,z\,d + e2^2\,z\,a - c\,a^2\,e$$
$$- 3\,e2\,a^2\,d + 2\,a^4\,d + e2^2\,d - e2^2\,a + 3\,e2\,a^3 - 3\,a^3\,z\,e2 + 2\,a^5\,z - 2\,a^5 + 3\,a^2\,z\,e2\,d$$

$$-c\,e2\,z\,e+c\,a^2\,z\,e+2\,c\,a^2\,z\,d+c\,e2\,z\,a-c\,e2\,z\,d),(-2\,a^4\,e-e2^2\,e+2\,c\,e2\,z\,e$$
$$-3\,c\,a^2\,z\,e-6\,a^2\,z\,e2\,e+3\,a^2\,z^2\,e2\,e-2\,c\,e2\,z^2\,e+3\,c\,a^2\,z^2\,e+4\,a^4\,z\,d+e2^2\,z^2\,a$$
$$+4\,a^3\,z\,e2+3\,e2\,a^2\,e-e2^2\,z^2\,d-e2^2\,z\,a+e2^2\,z\,d+2\,e2^2\,z\,e-e2^2\,z^2\,e+4\,a^4\,z\,e$$
$$-2\,a^4\,z^2\,e-4\,a^3\,z^2\,e2-z^2\,c^2\,e-4\,a^4\,z^2\,d+4\,a^5\,z^2-4\,a^5\,z-c\,e2\,z^2\,d+2\,c\,a^2\,z^2\,d$$
$$+c\,e2\,z^2\,a+4\,a^2\,z^2\,e2\,d-4\,a^2\,z\,e2\,d-2\,c\,a^3\,z^2)\Big/(-2\,c\,a^3\,z-z\,c^2\,e-2\,a^4\,z\,d$$
$$+c\,e2\,e-e2^2\,z\,d+e2^2\,z\,a-c\,a^2\,e-3\,e2\,a^2\,d+2\,a^4\,d+e2^2\,d-e2^2\,a+3\,e2\,a^3$$
$$-3\,a^3\,z\,e2+2\,a^5\,z-2\,a^5+3\,a^2\,z\,e2\,d-c\,e2\,z\,e+c\,a^2\,z\,e+2\,c\,a^2\,z\,d+c\,e2\,z\,a$$
$$-c\,e2\,z\,d)]$$

```
> indets(xS);
```

$$\{a, c, d, z, e2, e\}$$

```
> #xP:=unapply(xS,convert(indets(xS),list)):
  #xPO:=codegen[optimize](xP);
  #"Gespart: ",codegen[cost](xP)-codegen[cost](xPO);
```

Nebenrechnung 2

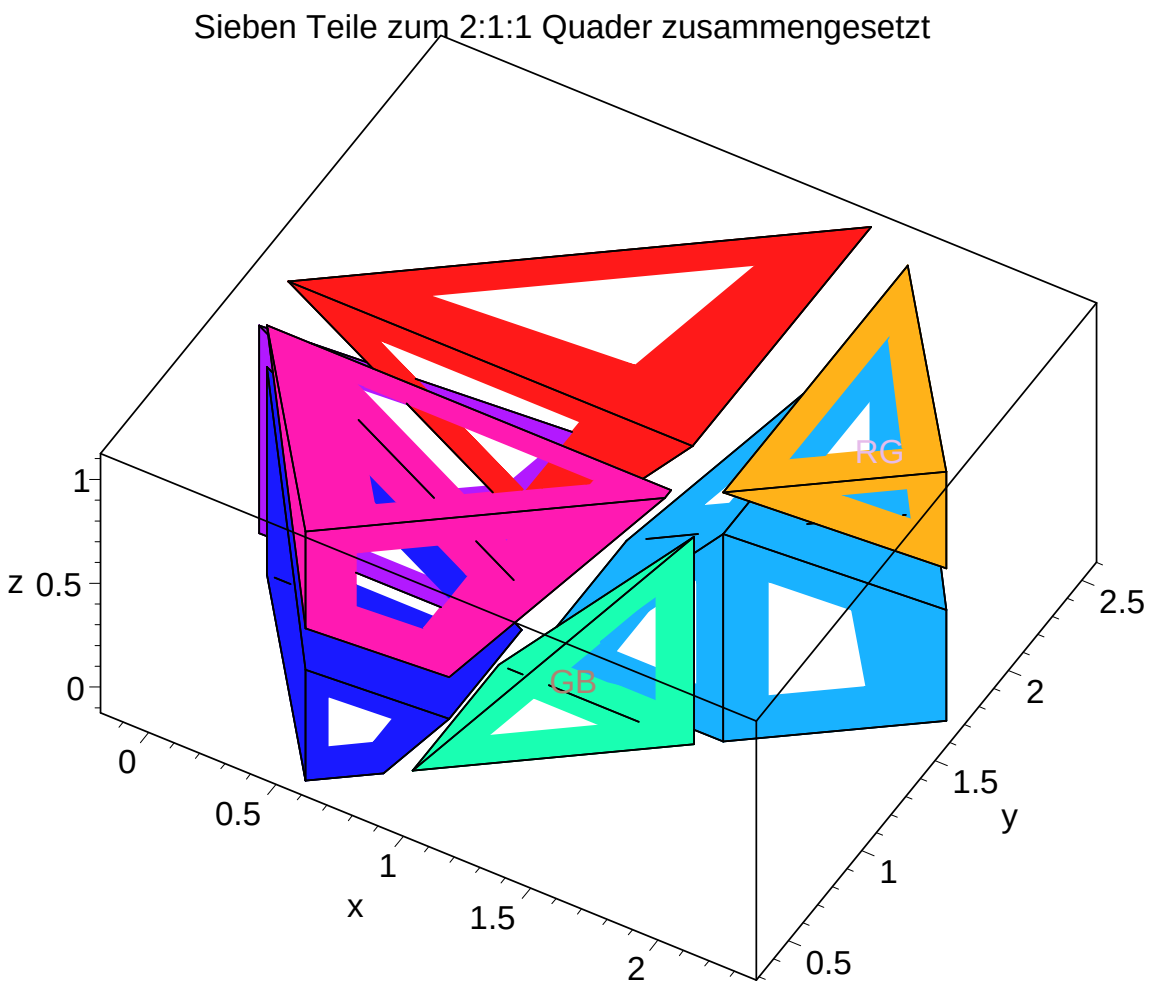
Da die wunderschöne Beobachtung nur an einer winzigen Verschiebung scheitert, will ich nun doch noch einen Versuch wagen, mit sieben Teilen auszukommen.
Dazu müssen die Teile GR, BG und GB verändert werden. Ich werde das Teil GR ganz und ein Scheibe des Teils GB an BG anheften. Das Teil GB wird etwas kleiner und seine Verschiebungsvektoren werden sich ändern. Wir müssen beachten, dass alle Angaben immer relativ zu der mittleren Situation des a^2:a:1-Quaders gemacht sind.

```
> cut_cube7 := proc(f, mv, S)
  local a, c, d, e, α, s, c2, d2, e2, A, y, B, z, C, RR, RG, RB, GR, GG, GB, BR, BG, BB, RX, GX,
  BX, XR, XG, XB, RX1, GX1, BX1, XR1, XG1, XB1, RX2, GX2, BX2, XR2, XG2, XB2, X0, Y0,
  Z0, X1, Y1, Z1, U, AU, L, t, NR1, NR2;
      a := 2^(1 / 3);
      c, d, e, α := rectangle_data(a^2, a, 1);
      s := sqrt(a^2 – 1);
      e2 := expand(a^2 – s);
      c2, d2 := op(expand((1 – s)*[a^2, 1] + s*[e2, 0]));
      A := [e2, e*e2 / (a^2), 0];
      expand(λ*[a^2, e, 1] + (1 – λ)*A – [c2, y, d2]);
      convert(%, set);
      solve(%);
      y := simplify(subs(%, y));
      B := [c2, y, d2];
      expand(λ*[0, 0, 1] + (1 – λ)*B – [c, d, z]);
      convert(%, set);
      solve(%);
      z := simplify(subs(%, z));
      C := [c, d, z];
      U := [0, a – e, 0];
      AU := simplify(expand(A + U));
      RR := polyhedron("RR", [[0, 0, 1], [a^2, 0, 1], [a^2, e, 1], [c2, 0, d2], B],
          color = RRcolor);
```

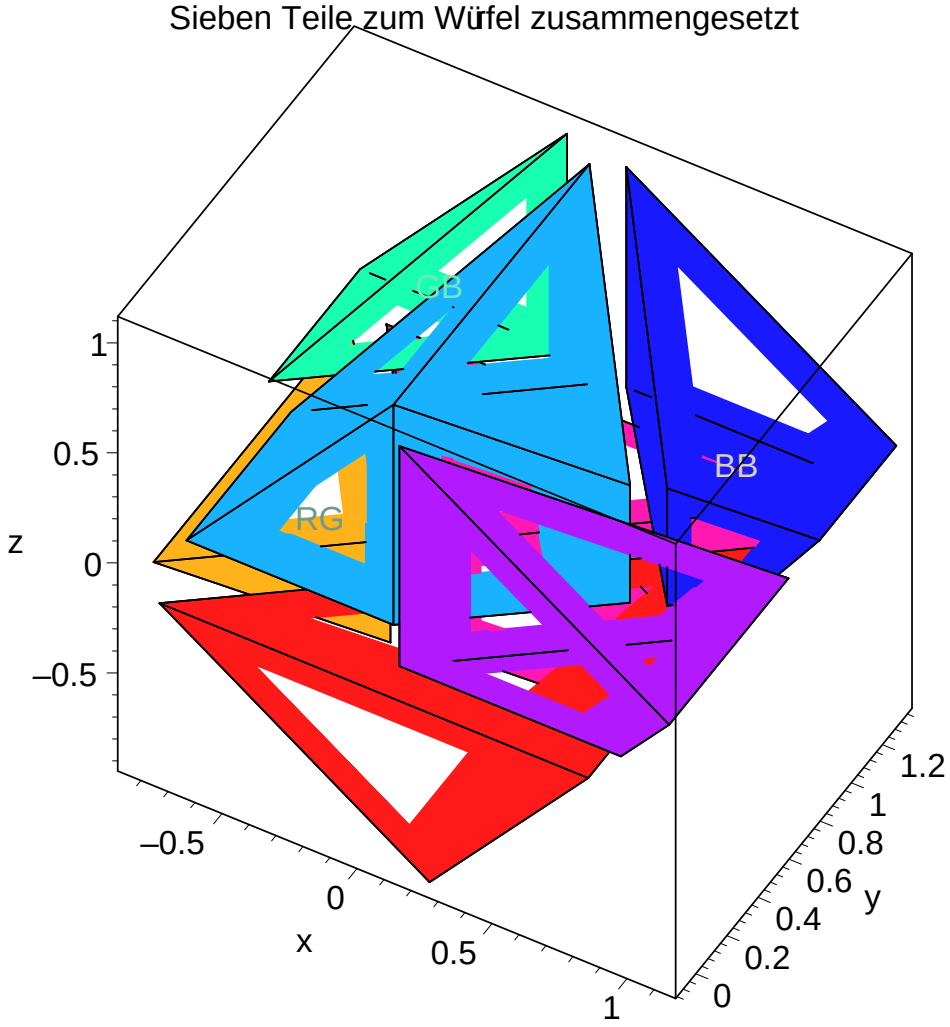
```
userinfo(4, delos, "RR", print(%));
RG := polyhedron("RG", [[0, 0, 1], [c, d, 1], [0, a, 1], C], color = RGcolor);
userinfo(4, delos, "RG", print(%));
RB := polyhedron("RB", [[0, a, 1], [c, d, 1], [a^2, e, 1], [a^2, a, 1], C, B, [c2, a, d2]],
      color = RBcolor);
userinfo(4, delos, "RB", print(%));
GR := NULL;
userinfo(4, delos, "GR", print(%));
GG := NULL;
GB := polyhedron("GB", [subs(s = evalf(s), AU), [a^2, a, 0], [e2, a, 0], [a^2, a, 1]],
      color = GBcolor);
userinfo(4, delos, "GB", print(%));
BR := polyhedron("BR", [[0, 0, 0], [0, 0, 1], [c2, 0, d2], [e2, 0, 0], A, B],
      color = BRcolor);
userinfo(4, delos, "BR", print(%));
BG := polyhedron("BG", [
      [0, 0, 0], [0, 0, 1], C, [c, d, 0], [0, a, 0], [0, a, 1], [e2 - a^2, 0, 0], AU - [a^2, 0, 0]],
      color = BGcolor);
userinfo(4, delos, "BG", print(%));
BB := polyhedron("BB", [[0, a, 0], [0, a, 1], C, [c, d, 0], A, [e2, a, 0], [c2, a, d2], B],
      color = BBcolor);
userinfo(4, delos, "BB", print(%));
XR := [0, a, 0];
XR1 := [a^2, a, 0];
XR2 := [a^2, e, 0];
XG := [a^2, e, 0];
XG1 := [a^2, e, 0] - [0, a, 0];
XG2 := [c, d, 0] - [0, a, 0];
XB := [0, 0, 0];
XB1 := XB;
XB2 := XB;
RX := [e2, 0, 0] - [a^2, 0, 1];
RX1 := [e2, 0, 0] - [0, 0, 1];
RX2 := [c2, 0, d2] - [0, 0, 1];
GX := -[a^2, 0, 0];
GX1 := -[a^2, 0, 1];
GX2 := [e2, 0, 0] - [a^2, 0, 1];
BX := [0, 0, 0];
BX1 := BX;
BX2 := BX;
X0 := A - [c, d, 0];
X0 := X0 / LinearAlgebra:-Norm(convert(X0, Vector), 2);
Y0 := [0, a, 0] - [c, d, 0];
Y0 := Y0 / LinearAlgebra:-Norm(convert(Y0, Vector), 2);
Z0 := C - [c, d, 0];
Z0 := Z0 / LinearAlgebra:-Norm(convert(Z0, Vector), 2);
```

```
X1 := [0, a, 1] – [c2, a, d2];
X1 := X1 / LinearAlgebra:-Norm(convert(X1, Vector), 2);
Y1 := [e2, a, 0] – [c2, a, d2];
Y1 := Y1 / LinearAlgebra:-Norm(convert(Y1, Vector), 2);
Z1 := B – [c2, a, d2];
Z1 := Z1 / LinearAlgebra:-Norm(convert(Z1, Vector), 2);
L := NULL;
NR1 := evalf([(2*e*c^2*z*e2 + 2*a^6*d*z + 2*c*e*a^2*e2 + e2^2*d*z*a^2
    – 2*a^5*c*z – 2*a^4*d*c – e*c^2*e2 – e2^2*d*a^2 + 3*e2*d*a^4
    – 3*e2*d*a^4*z + e*c^2*a^2 – 2*a^2*d*c^2*z + 2*a^4*d*c*z – 3*e2*d*c*z*a^2
    – e2^2*d*c + 2*a^3*c^2*z – 3*e2*a^3*c + c*e2^2*e*z – e2*a*c^2*z
    – e2^2*a*c*z + 3*e2*a^3*c*z + e2*d*c^2*z + 3*e2*d*c*a^2 + e2^2*d*c*z
    + 2*a^5*c – 2*c*e*a^2*z*e2 – 2*a^6*d + c*e*a^4*z – 2*e*c^2*z*a^2 + e*c^3*z
    – c*e*a^4 – c*e2^2*e + e2^2*a*c + 3*e2*a^5*z – e2^2*a^3*z – 3*e2*a^5
    + e2^2*a^3 – 2*a^7*z + 2*a^7) / (–c*e2*e – 2*z*d*c*a^2 + 2*z*d*a^4
    – 2*a^4*d – e2^2*d + e2^2*d*z + 3*e2*d*a^2 + z*d*c*e2 – 3*z*d*e2*a^2
    + 2*z*a^3*c – 2*a^5*z – z*a*c*e2 + 3*e2*a^3*z + e2^2*a – 3*e2*a^3
    – e2^2*a*z + 2*a^5 + e*c*z*e2 + e*c^2*z – e*c*z*a^2 + c*e*a^2), (–a*e*c*z*e2
    – a*z*d*c*e2 – 3*e2*e*a^3 + 2*z*d*c*a^3 + 3*e2*e*z*a^3 + 3*z*d*e2*a^3
    – 2*z*d*a^5 – 2*e*a^5*z + 2*e*a^5 – 3*z*d^2*a^2*e2 – 3*z*d*e*a^2*e2
    + 3*d*e*a^2*e2 + 2*z*d^2*a^4 – 2*d*e*a^4 + 2*z*d*e*a^4 – 2*e*z*d*c*a^2
    + 2*z*d*c*e2*e + e*z*a^3*c + e2^2*d*e*z – 2*d^2*a^4 – 3*d*e2*a^3
    + 3*d^2*a^2*e2 + 2*d*a^5 – e2^2*d*e – e2^2*d*a*z + e2^2*d*a – a*e*z*e2^2
    + a*e*e2^2 + e*d*c*a^2 – d*c*e2*e – e2^2*d^2 – 2*z*d^2*c*a^2 + z*d^2*c*e2
    + e2^2*d^2*z + d*e*c^2*z) / (–c*e2*e – 2*z*d*c*a^2 + 2*z*d*a^4 – 2*a^4*d
    – e2^2*d + e2^2*d*z + 3*e2*d*a^2 + z*d*c*e2 – 3*z*d*e2*a^2 + 2*z*a^3*c
    – 2*a^5*z – z*a*c*e2 + 3*e2*a^3*z + e2^2*a – 3*e2*a^3 – e2^2*a*z + 2*a^5
    + e*c*z*e2 + e*c^2*z – e*c*z*a^2 + c*e*a^2), (–a*z^2*e2^2 – 4*e2*a^3*z
    + 4*a^5*z + c^2*e*z^2 + 2*c*e*z^2*e2 + 4*z^2*d*a^4 + 6*e2*e*z*a^2
    – 3*e*z^2*e2*a^2 + 2*e*a^4 + 4*z*d*e2*a^2 – 4*z^2*d*e2*a^2
    – 3*c*e*z^2*a^2 – 4*z*d*a^4 + z^2*d*c*e2 – 2*z^2*d*c*a^2 + 2*z^2*a^3*c
    – 4*e*a^4*z + 2*e*z^2*a^4 – 2*e*c*z*e2 + 3*e*c*z*a^2 – z^2*a*c*e2
    – 3*e2*e*a^2 – 4*z^2*a^5 + e2^2*e – e2^2*d*z – 2*e2^2*e*z + e2^2*a*z
    + z^2*e2^2*d + e*z^2*e2^2 + 4*z^2*e2*a^3) / (–c*e2*e – 2*z*d*c*a^2
    + 2*z*d*a^4 – 2*a^4*d – e2^2*d + e2^2*d*z + 3*e2*d*a^2 + z*d*c*e2
    – 3*z*d*e2*a^2 + 2*z*a^3*c – 2*a^5*z – z*a*c*e2 + 3*e2*a^3*z + e2^2*a
    – 3*e2*a^3 – e2^2*a*z + 2*a^5 + e*c*z*e2 + e*c^2*z – e*c*z*a^2 + c*e*a^2)]);
NR2 := evalf((–a*c – d2*e2*a – c2*z*a + d2*a^3 – d2*d*a^2 – a^3 + d2*e2*d
    + d2*a*c + c2*d + d*a^2 + e2*a – e2*d) / (c2*(–a + d)));
for t from 0 by 1 / S to 1 do L := L, display([translate(RR, mv(f(1, t), XR, XR1, XR2,
    [0, 0, 0], RX2, RX1, RX, Y0 + Z0, [0, –1, 1], –X1 + Y1 + Z1, XR + [0, e, 0],
    XR + A – [a^2, 0, 1])), translate(RG, mv(f(3, t), XG, XG1, XG2, [0, 0, 0], RX2, RX1,
    RX, X0 – Y0 + Z0, [–1, 0, 1], Y1, NR1 – C, RX + [0, 0, 1 – z])), translate(RB, mv(
    f(2, t), XB, XB1, XB2, [0, 0, 0], RX2, RX1, RX, –X0 – Y0 + Z0, [0, 1, 1],
    –X1 + Y1 – Z1, XB + [0, a, 0], XB + [0, a, 0] + A – [a^2, e, 1])), translate(GB, mv(
    f(6, t), XB – U, XB1 – U, XB2 – U, [0, 0, 0] – U, GX2 – U, GX1, GX, –Y0 – Z0,
```

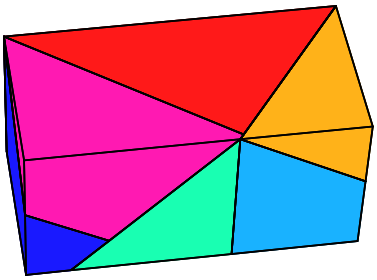
```
            [1, 1, –1], X1 – Z1, XB – U + [e2, a, 0] – A, XB – U + [0, a, 0] – A)), translate(BR,
            mv(f(4, t), XR, XR1, XR2, [0, 0, 0], BX2, BX1, BX, –X0 + Y0 – Z0, [–1, –1, –1],
            –X1 – Y1 + Z1, XR + [0, 0, 1], C – A)), translate(BG, mv(f(5, t), XG, XG1, XG2,
            [0, 0, 0], BX2, BX1, BX, X0 – Y0 – Z0, [–1, 0, –1], –Y1, [a^2, a, 0],
            [a*c / (d – a), a, 0])), translate(BB, mv(f(7, t), XB, XB1, XB2, [0, 0, 0], BX2, BX1,
            BX, –X0 – Y0 – Z0, [0, 1, –1], –X1 – Y1 – Z1, [0, 0, 0], [0, 0, 0])), NULL])
        end do;
        display([L], insequence = true, scaling = CONSTRAINED, orientation = [–60, 45],
            args[4 .. –1])
    end proc
> f7e := proc(n, t) max(0, min(1, 6*t – n + 1)) end proc
> mv7_3l := proc(t, A0, A1, A2, A3)
            if t < 1 / 3 then mv0l_1(3*t, A0, A1)
            elif t < 2 / 3 then mv0l_1(3*t – 1, A1, A2)
            else mv0l_1(3*t – 2, A2, A3)
            end if;
            op(evalf(%))
        end proc;
    mv7_4l := proc(t, A0, A1, A2, A3, A4)
            if t < 1 / 4 then mv0l_1(4*t, A0, A1)
            elif t < 1 / 2 then mv0l_1(4*t – 1, A1, A2)
            elif t < 3 / 4 then mv0l_1(4*t – 2, A2, A3)
            else mv0l_1(4*t – 3, A3, A4)
            end if;
            op(evalf(%))
        end proc;
    mv7l := proc(t, A, B1, B2, B3, B4, B5, C, DA, DB, DC, Z1, Z2, Z3)
            if nargs < 13 then mv1(args)
            elif nargs < 14 then mv7_3l(t, A, Z1, Z2, C)
            else mv7_4l(t, A, Z1, Z2, Z3, C)
            end if
        end proc;
    mv7c := proc(t, A, B1, B2, B3, B4, B5, C, DA, DB, DC, Z1, Z2, Z3)
            if nargs < 13 then mv1(args)
            else mv0c_3aInterp(evalf(A), evalf(Z1), evalf(Z2), evalf(C))(t); op(evalf(%))
            end if
        end proc
Hiermit können wir gut testen.
> # cut_cube7( f7e, mv7l, 18, labels=[x,y,z] );
> # display(subs( {TEXT=(x->NULL)},cut_cube7(f7e,mv7l,6,labels
  = [x, y, z],projection = normal)),insequence=false );
> mv := proc(t, A, B1, B2, B3, B4, B5, C, DA, DB, DC) A + t*DA; op(evalf(%)) end proc;
cut_cube7((n, t) → .1, mv, .5, labels = [x, y, z], axes = boxed, orientation = [–60, 45],
    title = "Sieben Teile zum 2:1:1 Quader zusammengesetzt");
mycutout(%, .6)
```




```
> mv := proc(t, A, B1, B2, B3, B4, B5, C, DA, DB, DC) C + t*DC; op(evalf(%)) end proc;
  cut_cube7((n, t) → .1, mv, .5, labels = [x, y, z], axes = boxed, orientation = [–60, 45],
        title = "Sieben Teile zum Würfel zusammengesetzt");
  mycutout(%, .6)
```



> cut_cube7(*f7e@looptime2*, *mv7c*, 180, *labels* = [*x*, *y*, *z*], *projection* = *normal*)



Den Film nochmal in einem getrennten Fenster plotten, um ihn getrennt zu speichern.

```
> plotsetup(window);
  cut_cube7(f7e@looptime2, mv7c, 360, labels = [x, y, z], projection = normal);
  plotsetup(default)
```

```
> B1:='B1';
```

$$B1 := B1$$

```
> tracelast;
  RTableQueryImplementation:-query called with arguments: 14119588, num_dims
  #(RTableQueryImplementation:-query,20): error
Error, (in RTableQueryImplementation:-query) invalid rtable handle 14119588

  locals defined as: rt = rt, q = q
>
```

Gitter?

Viele Polyederzerlegungen stammen von zwei überlagerten Kachelungen. Wir wollen sehen, ob das hier auch so ist. Vorweg: die Quadrat-Rechteck-Verwandlungen, die wir verwendet

haben, sind von diesem Typ. Aber das 'Umleimen' hat vielleicht etwas kaputt gemacht. Die Argumente S0, S1 und T geben an, wie oft Würfel, Altar und Gitterpunkte gezeichnet werden sollen.

```
> cut_cube7_fill := proc(S0, S1, T)
  local a, c, d, e, α, s, c2, d2, e2, A, y, B, z, C, RR, RG, RB, GR, GG, GB, BR, BG, BB, RRdata,
  RGdata, RBdata, GRdata, GGdata, GBdata, BRdata, BGdata, BBdata, RX, GX, BX, XR, XG,
  XB, RX1, GX1, BX1, XR1, XG1, XB1, RX2, GX2, BX2, XR2, XG2, XB2, X0, Y0, Z0, X1, Y1, Z1,
  U, AU, L, t, NR1, NR2, Q, i, j, k;
      a := 2^(1 / 3);
      c, d, e, α := rectangle_data(a^2, a, 1);
      s := sqrt(a^2 – 1);
      e2 := expand(a^2 – s);
      c2, d2 := op(expand((1 – s)*[a^2, 1] + s*[e2, 0]));
      A := [e2, e*e2 / (a^2), 0];
      expand(λ*[a^2, e, 1] + (1 – λ)*A – [c2, y, d2]);
      convert(%, set);
      solve(%);
      y := simplify(subs(%, y));
      B := [c2, y, d2];
      expand(λ*[0, 0, 1] + (1 – λ)*B – [c, d, z]);
      convert(%, set);
      solve(%);
      z := simplify(subs(%, z));
      C := [c, d, z];
      U := [0, a – e, 0];
      AU := simplify(expand(A + U));
      RR := polyhedron("RR", [[0, 0, 1], [a^2, 0, 1], [a^2, e, 1], [c2, 0, d2], B],
          color = RRcolor);
      userinfo(4, delos, "RR", print(%));
      RG := polyhedron("RG", [[0, 0, 1], [c, d, 1], [0, a, 1], C], color = RGcolor);
      userinfo(4, delos, "RG", print(%));
      RB := polyhedron("RB", [[0, a, 1], [c, d, 1], [a^2, e, 1], [a^2, a, 1], C, B, [c2, a, d2]],
          color = RBcolor);
      userinfo(4, delos, "RB", print(%));
      GR := NULL;
      userinfo(4, delos, "GR", print(%));
      GG := NULL;
      GB := polyhedron("GB", [subs(s = evalf(s), AU), [a^2, a, 0], [e2, a, 0], [a^2, a, 1]],
          color = GBcolor);
      userinfo(4, delos, "GB", print(%));
      BR := polyhedron("BR", [[0, 0, 0], [0, 0, 1], [c2, 0, d2], [e2, 0, 0], A, B],
          color = BRcolor);
      userinfo(4, delos, "BR", print(%));
      BG := polyhedron("BG", [
          [0, 0, 0], [0, 0, 1], C, [c, d, 0], [0, a, 0], [0, a, 1], [e2 – a^2, 0, 0], AU – [a^2, 0, 0]],
          color = BGcolor);
```

```
userinfo(4, delos, "BG", print(%));
BB := polyhedron("BB", [[0, a, 0], [0, a, 1], C, [c, d, 0], A, [e2, a, 0], [c2, a, d2], B],
      color = BBcolor);
userinfo(4, delos, "BB", print(%));
XR := [0, a, 0];
XR1 := [a^2, a, 0];
XR2 := [a^2, e, 0];
XG := [a^2, e, 0];
XG1 := [a^2, e, 0] – [0, a, 0];
XG2 := [c, d, 0] – [0, a, 0];
XB := [0, 0, 0];
XB1 := XB;
XB2 := XB;
RX := [e2, 0, 0] – [a^2, 0, 1];
RX1 := [e2, 0, 0] – [0, 0, 1];
RX2 := [c2, 0, d2] – [0, 0, 1];
GX := –[a^2, 0, 0];
GX1 := –[a^2, 0, 1];
GX2 := [e2, 0, 0] – [a^2, 0, 1];
BX := [0, 0, 0];
BX1 := BX;
BX2 := BX;
X0 := A – [c, d, 0];
X0 := X0 / LinearAlgebra:-Norm(convert(X0, Vector), 2);
Y0 := [0, a, 0] – [c, d, 0];
Y0 := Y0 / LinearAlgebra:-Norm(convert(Y0, Vector), 2);
Z0 := C – [c, d, 0];
Z0 := Z0 / LinearAlgebra:-Norm(convert(Z0, Vector), 2);
X1 := [0, a, 1] – [c2, a, d2];
X1 := X1 / LinearAlgebra:-Norm(convert(X1, Vector), 2);
Y1 := [e2, a, 0] – [c2, a, d2];
Y1 := Y1 / LinearAlgebra:-Norm(convert(Y1, Vector), 2);
Z1 := B – [c2, a, d2];
Z1 := Z1 / LinearAlgebra:-Norm(convert(Z1, Vector), 2);
NR1 := evalf([(2*e*c^2*z*e2 + 2*a^6*d*z + 2*c*e*a^2*e2 + e2^2*d*z*a^2
      – 2*a^5*c*z – 2*a^4*d*c – e*c^2*e2 – e2^2*d*a^2 + 3*e2*d*a^4
      – 3*e2*d*a^4*z + e*c^2*a^2 – 2*a^2*d*c^2*z + 2*a^4*d*c*z – 3*e2*d*c*z*a^2
      – e2^2*d*c + 2*a^3*c^2*z – 3*e2*a^3*c + c*e2^2*e*z – e2*a*c^2*z
      – e2^2*a*c*z + 3*e2*a^3*c*z + e2*d*c^2*z + 3*e2*d*c*a^2 + e2^2*d*c*z
      + 2*a^5*c – 2*c*e*a^2*z*e2 – 2*a^6*d + c*e*a^4*z – 2*e*c^2*z*a^2 + e*c^3*z
      – c*e*a^4 – c*e2^2*e + e2^2*a*c + 3*e2*a^5*z – e2^2*a^3*z – 3*e2*a^5
      + e2^2*a^3 – 2*a^7*z + 2*a^7) / (–c*e2*e – 2*z*d*c*a^2 + 2*z*d*a^4
      – 2*a^4*d – e2^2*d + e2^2*d*z + 3*e2*d*a^2 + z*d*c*e2 – 3*z*d*e2*a^2
      + 2*z*a^3*c – 2*a^5*z – z*a*c*e2 + 3*e2*a^3*z + e2^2*a – 3*e2*a^3
      – e2^2*a*z + 2*a^5 + e*c*z*e2 + e*c^2*z – e*c*z*a^2 + c*e*a^2), (–a*e*c*z*e2
      – a*z*d*c*e2 – 3*e2*e*a^3 + 2*z*d*c*a^3 + 3*e2*e*z*a^3 + 3*z*d*e2*a^3
```

```
        – 2*z*d*a^5 – 2*e*a^5*z + 2*e*a^5 – 3*z*d^2*a^2*e2 – 3*z*d*e*a^2*e2
        + 3*d*e*a^2*e2 + 2*z*d^2*a^4 – 2*d*e*a^4 + 2*z*d*e*a^4 – 2*e*z*d*c*a^2
        + 2*z*d*c*e2*e + e*z*a^3*c + e2^2*d*e*z – 2*d^2*a^4 – 3*d*e2*a^3
        + 3*d^2*a^2*e2 + 2*d*a^5 – e2^2*d*e – e2^2*d*a*z + e2^2*d*a – a*e*z*e2^2
        + a*e*e2^2 + e*d*c*a^2 – d*c*e2*e – e2^2*d^2 – 2*z*d^2*c*a^2 + z*d^2*c*e2
        + e2^2*d^2*z + d*e*c^2*z) / (–c*e2*e – 2*z*d*c*a^2 + 2*z*d*a^4 – 2*a^4*d
        – e2^2*d + e2^2*d*z + 3*e2*d*a^2 + z*d*c*e2 – 3*z*d*e2*a^2 + 2*z*a^3*c
        – 2*a^5*z – z*a*c*e2 + 3*e2*a^3*z + e2^2*a – 3*e2*a^3 – e2^2*a*z + 2*a^5
        + e*c*z*e2 + e*c^2*z – e*c*z*a^2 + c*e*a^2), (–a*z^2*e2^2 – 4*e2*a^3*z
        + 4*a^5*z + c^2*e*z^2 + 2*c*e*z^2*e2 + 4*z^2*d*a^4 + 6*e2*e*z*a^2
        – 3*e*z^2*e2*a^2 + 2*e*a^4 + 4*z*d*e2*a^2 – 4*z^2*d*e2*a^2
        – 3*c*e*z^2*a^2 – 4*z*d*a^4 + z^2*d*c*e2 – 2*z^2*d*c*a^2 + 2*z^2*a^3*c
        – 4*e*a^4*z + 2*e*z^2*a^4 – 2*e*c*z*e2 + 3*e*c*z*a^2 – z^2*a*c*e2
        – 3*e2*e*a^2 – 4*z^2*a^5 + e2^2*e – e2^2*d*z – 2*e2^2*e*z + e2^2*a*z
        + z^2*e2^2*d + e*z^2*e2^2 + 4*z^2*e2*a^3) / (–c*e2*e – 2*z*d*c*a^2
        + 2*z*d*a^4 – 2*a^4*d – e2^2*d + e2^2*d*z + 3*e2*d*a^2 + z*d*c*e2
        – 3*z*d*e2*a^2 + 2*z*a^3*c – 2*a^5*z – z*a*c*e2 + 3*e2*a^3*z + e2^2*a
        – 3*e2*a^3 – e2^2*a*z + 2*a^5 + e*c*z*e2 + e*c^2*z – e*c*z*a^2 + c*e*a^2)]);
NR2 := evalf((–a*c – d2*e2*a – c2*z*a + d2*a^3 – d2*d*a^2 – a^3 + d2*e2*d
        + d2*a*c + c2*d + d*a^2 + e2*a – e2*d) / (c2*(–a + d)));
RRdata := XR, XR1, XR2, [0, 0, 0], RX2, RX1, RX, Y0 + Z0, [0, –1, 1], –X1 + Y1 + Z1,
        XR + [0, e, 0], XR + A – [a^2, 0, 1];
RGdata := XG, XG1, XG2, [0, 0, 0], RX2, RX1, RX, X0 – Y0 + Z0, [–1, 0, 1], Y1,
        NR1 – C, RX + [0, 0, 1 – z];
RBdata := XB, XB1, XB2, [0, 0, 0], RX2, RX1, RX, –X0 – Y0 + Z0, [0, 1, 1],
        –X1 + Y1 – Z1, XB + [0, a, 0], XB + [0, a, 0] + A – [a^2, e, 1];
GBdata := XB – U, XB1 – U, XB2 – U, [0, 0, 0] – U, GX2 – U, GX1, GX, –Y0 – Z0,
        [1, 1, –1], X1 – Z1, XB – U + [e2, a, 0] – A, XB – U + [0, a, 0] – A;
BRdata := XR, XR1, XR2, [0, 0, 0], BX2, BX1, BX, –X0 + Y0 – Z0, [–1, –1, –1],
        –X1 – Y1 + Z1, XR + [0, 0, 1], C – A;
BGdata := XG, XG1, XG2, [0, 0, 0], BX2, BX1, BX, X0 – Y0 – Z0, [–1, 0, –1], –Y1,
        [a^2, a, 0], [a*c / (d – a), a, 0];
BBdata := XB, XB1, XB2, [0, 0, 0], BX2, BX1, BX, –X0 – Y0 – Z0, [0, 1, –1],
        –X1 – Y1 – Z1, [0, 0, 0], [0, 0, 0];
for i from 0 to 1 do Q[i] := display([
        translate(RR, op(RRdata[1] + i*(RRdata[7] – RRdata[1]))),
        translate(RG, op(RGdata[1] + i*(RGdata[7] – RGdata[1]))),
        translate(RB, op(RBdata[1] + i*(RBdata[7] – RBdata[1]))),
        translate(GB, op(GBdata[1] + i*(GBdata[7] – GBdata[1]))),
        translate(BR, op(BRdata[1] + i*(BRdata[7] – BRdata[1]))),
        translate(BG, op(BGdata[1] + i*(BGdata[7] – BGdata[1]))),
        translate(BB, op(BBdata[1] + i*(BBdata[7] – BBdata[1]))), NULL])
end do;
if false then
        L := subs({a = ω, a^2 = ω^2},
                simplify(Transpose(Matrix(expand([2*X0, a*Z1, RX – XR])))));
```

```
        L := subs(ω = RootOf(_Z^3 – 2), L);
        L := map(((simplify@rationalize)@expand)@@3, L);
        L := subs(RootOf(_Z^3 – 2) = ω, L);
        print(L = eval(L, ω = evalf(a)))
    end if;
    L := NULL;
    Q[2] := translate(BR, op(BRdata[1] + i*(BRdata[7] – BRdata[1])));
    for i from –S0 to S0 do for j from –S0 to S0 do for k from –S0 to S0 do
            s := evalf(i*2*X0 + j*a*Z1 + k*(RX – XR)); L := L, translate(Q[0], op(s))
        end do
    end do
    end do;
    for i from –S1 to S1 do for j from –S1 to S1 do for k from –S1 to S1 do
            s := evalf(i*2*X0 + j*a*Z1 + k*(RX – XR)); L := L, translate(Q[1], op(s))
        end do
    end do
    end do;
    for i from –T to T do for j from –T to T do for k from –T to T do
            s := evalf(i*2*X0 + j*a*Z1 + k*(RRdata[7] – RRdata[1]));
            if true or evalf(convert(map(x → x^2, s), `+`) < 5*a^2*(S + 1)^2) then
                L := L, display(plottools[octahedron](s, .1))
            end if
        end do
    end do
    end do;
    display([L], insequence = false, scaling = CONSTRAINED, orientation = [–60, 45],
        args[4 .. –1])
end proc
```

Nein, doch nicht, es ist alles ok.

Gitterbasis.

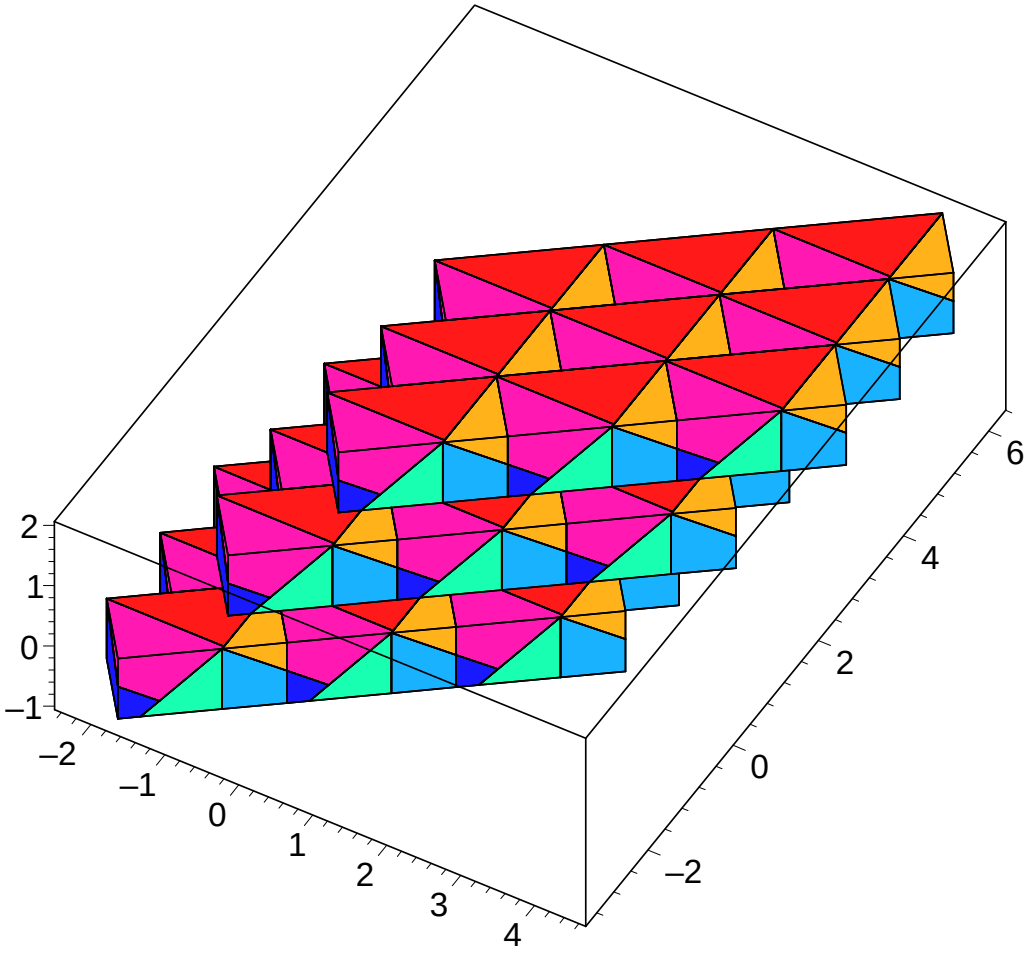
```
> cut_cube7_fillbasis := proc()
  local a, c, d, e, α, s, c2, d2, e2, A, y, B, z, C, RR, RG, RB, GR, GG, GB, BR, BG, BB, RRdata,
  RGdata, RBdata, GRdata, GGdata, GBdata, BRdata, BGdata, BBdata, RX, GX, BX, XR, XG,
  XB, RX1, GX1, BX1, XR1, XG1, XB1, RX2, GX2, BX2, XR2, XG2, XB2, X0, Y0, Z0, X1, Y1, Z1,
  U, AU, L, t, NR1, NR2, Q, i, j, k;
      a := 2^(1 / 3);
      c, d, e, α := rectangle_data(a^2, a, 1);
      s := sqrt(a^2 – 1);
      e2 := expand(a^2 – s);
      c2, d2 := op(expand((1 – s)*[a^2, 1] + s*[e2, 0]));
      A := [e2, e*e2 / (a^2), 0];
      expand(λ*[a^2, e, 1] + (1 – λ)*A – [c2, y, d2]);
      convert(%, set);
      solve(%);
      y := simplify(subs(%, y));
      B := [c2, y, d2];
```

```
expand(λ*[0, 0, 1]+(1−λ)*B−[c, d, z]);
convert(%, set);
solve(%);
z := simplify(subs(%, z));
C := [c, d, z];
U := [0, a−e, 0];
AU := simplify(expand(A+U));
RR := polyhedron("RR", [[0, 0, 1], [a^2, 0, 1], [a^2, e, 1], [c2, 0, d2], B],
      color=RRcolor);
userinfo(4, delos, "RR", print(%));
RG := polyhedron("RG", [[0, 0, 1], [c, d, 1], [0, a, 1], C], color=RGcolor);
userinfo(4, delos, "RG", print(%));
RB := polyhedron("RB", [[0, a, 1], [c, d, 1], [a^2, e, 1], [a^2, a, 1], C, B, [c2, a, d2]],
      color=RBcolor);
userinfo(4, delos, "RB", print(%));
GR := NULL;
userinfo(4, delos, "GR", print(%));
GG := NULL;
GB := polyhedron("GB", [subs(s=evalf(s), AU), [a^2, a, 0], [e2, a, 0], [a^2, a, 1]],
      color=GBcolor);
userinfo(4, delos, "GB", print(%));
BR := polyhedron("BR", [[0, 0, 0], [0, 0, 1], [c2, 0, d2], [e2, 0, 0], A, B],
      color=BRcolor);
userinfo(4, delos, "BR", print(%));
BG := polyhedron("BG", [
      [0, 0, 0], [0, 0, 1], C, [c, d, 0], [0, a, 0], [0, a, 1], [e2−a^2, 0, 0], AU−[a^2, 0, 0]],
      color=BGcolor);
userinfo(4, delos, "BG", print(%));
BB := polyhedron("BB", [[0, a, 0], [0, a, 1], C, [c, d, 0], A, [e2, a, 0], [c2, a, d2], B],
      color=BBcolor);
userinfo(4, delos, "BB", print(%));
XR := [0, a, 0];
XR1 := [a^2, a, 0];
XR2 := [a^2, e, 0];
XG := [a^2, e, 0];
XG1 := [a^2, e, 0]−[0, a, 0];
XG2 := [c, d, 0]−[0, a, 0];
XB := [0, 0, 0];
XB1 := XB;
XB2 := XB;
RX := [e2, 0, 0]−[a^2, 0, 1];
RX1 := [e2, 0, 0]−[0, 0, 1];
RX2 := [c2, 0, d2]−[0, 0, 1];
GX := −[a^2, 0, 0];
GX1 := −[a^2, 0, 1];
GX2 := [e2, 0, 0]−[a^2, 0, 1];
```

```
        BX := [0, 0, 0];
        BX1 := BX;
        BX2 := BX;
        X0 := A – [c, d, 0];
        X0 := X0 / LinearAlgebra:-Norm(convert(X0, Vector), 2);
        Y0 := [0, a, 0] – [c, d, 0];
        Y0 := Y0 / LinearAlgebra:-Norm(convert(Y0, Vector), 2);
        Z0 := C – [c, d, 0];
        Z0 := Z0 / LinearAlgebra:-Norm(convert(Z0, Vector), 2);
        X1 := [0, a, 1] – [c2, a, d2];
        X1 := X1 / LinearAlgebra:-Norm(convert(X1, Vector), 2);
        Y1 := [e2, a, 0] – [c2, a, d2];
        Y1 := Y1 / LinearAlgebra:-Norm(convert(Y1, Vector), 2);
        Z1 := B – [c2, a, d2];
        Z1 := Z1 / LinearAlgebra:-Norm(convert(Z1, Vector), 2);
        L := subs({a = ω, a^2 = ω^2},
             simplify(Transpose(Matrix(expand([2*X0, a*Z1, RX – XR])))));
        L := subs(ω = RootOf(_Z^3 – 2), L);
        L := map(((simplify@rationalize)@expand)@@3, L);
        L := subs(RootOf(_Z^3 – 2) = ω, L);
        ω = a, L, simplify(Matrix(expand([X0, Y0, Z0]), scan = columns)),
             simplify(Matrix(expand([X1, Y1, Z1]), scan = columns))
    end proc
```

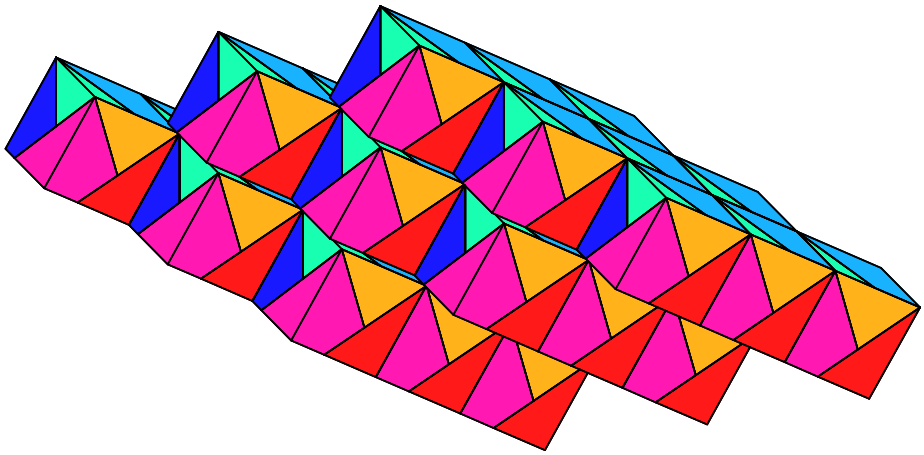
Nur Altare...

```
> eval( cut_cube7_fill(1,-1,-1), TEXT=(()->NULL) );
```



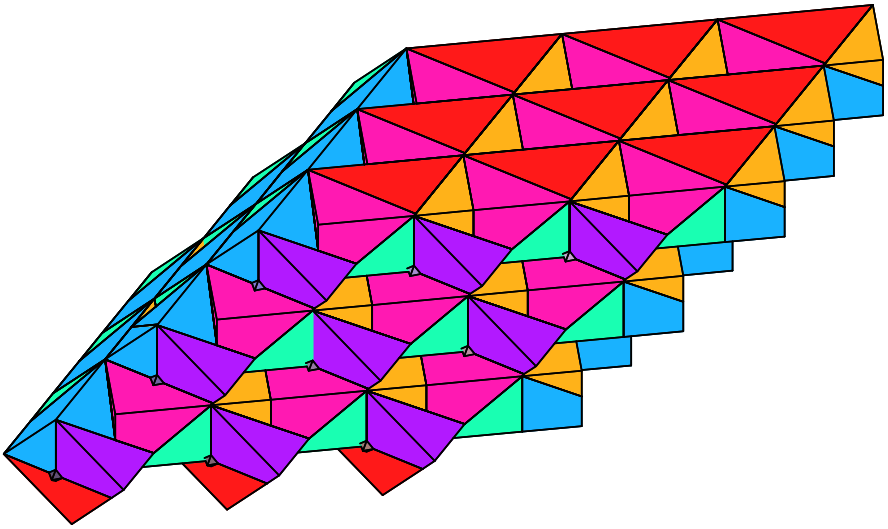
Nur Würfel...

```
> eval( cut_cube7_fill(-1,1,-1), TEXT=(()->NULL) );
```



Alles miteinander...

```
> eval( cut_cube7_fill(1,1,1), TEXT=(()->NULL) );
```



Machen wir die Probe...
(Da beim Zeichnen in Fließkommazahlen umgewandelt wurde, runden wir einfach nochmal, ur aus Fast-Gleichheit wieder Gleichheit zu machen.)
Wenn wir genug Würfel stapeln, sollten alle Teile von Altaren überdeckt sein.

```
> Altare:=eval( cut_cube7_fill(1/2,-1,-1), TEXT=(()->NULL) ):
  Wuerfel:=eval( cut_cube7_fill(-1,3/2,-1), TEXT=(()->NULL) ):
  A:=eval( convert(Altare,set),  POLYGONS=proc(x)
  POLYGONS(evalf(x,6),args[2..-1]) end proc):
  W:=eval( convert(Wuerfel,set), POLYGONS=proc(x)
  POLYGONS(evalf(x,6),args[2..-1]) end proc):
  display( op(A minus W), scaling=constrained,
  orientation=[148,123] );
```

Wir erwarten ein ’no object to display’!

```
Error, (in display) no object to display
```

Wenn wir genug Altare stapeln, sollten alle Teile von Würfeln überdeckt sein.

```
> Altare:=eval( cut_cube7_fill(3/2,-1,-1), TEXT=(()->NULL) ):
  Wuerfel:=eval( cut_cube7_fill(-1,1/2,-1), TEXT=(()->NULL) ):
  A:=eval( convert(Altare,set),  POLYGONS=proc(x)
```

```
  POLYGONS(evalf(x,6),args[2..-1]) end proc):
  W:=eval( convert(Wuerfel,set), POLYGONS=proc(x)
  POLYGONS(evalf(x,6),args[2..-1]) end proc):
  display( op(W minus A), scaling=constrained,
  orientation=[148,123] );
```

Wir erwarten ein 'no object to display'!

```
Error, (in display) no object to display
```

Untersuchen wir noch die Gitterbasis:

```
> S,M,B0,B1:=cut_cube7_fillbasis():
  M:=subs(S,M);
```

$$M := \left[\sqrt{-4\,2^{(1/3)}\sqrt{2^{(2/3)}-1}+7-4\sqrt{2^{(2/3)}-1}}\,(2\,2^{(2/3)}+\sqrt{2^{(2/3)}-1}\,2^{(2/3)}+2\sqrt{2^{(2/3)}-1}),\ 0,\ -\sqrt{2^{(2/3)}-1}\right]$$

$$\left[\sqrt{-4\,2^{(1/3)}\sqrt{2^{(2/3)}-1}+7-4\sqrt{2^{(2/3)}-1}}\,(2\sqrt{2^{(2/3)}-1}\,2^{(2/3)}+2\,2^{(1/3)}+2^{(2/3)}-2),\ -2^{(1/3)},\ -2^{(1/3)}\right]$$

$$[0,\ 0,\ -1]$$

Vereinfachungsversuche

Es hat eine Weile gedauert, diese Vereinfachung herauszubekommen...

```
> N:=map( x->expand(x^2)^(1/2)*signum(x), M):
```

Merken wir uns die Vereinfachungsfunktion für später...

```
> mysimplify := proc(x)
      if type(x, '+') then map(procname, x); simplify(eval(%))
      else simplify(expand(rationalize(expand(x^2))))^(1 / 2)*signum(x)
      end if;
      (simplify(%) assuming real)
  end proc
```

Noch allgemeiner funktioniert das hier: Hier wird versucht, die Eingabe x als algebraische Zahl vom Grad höchstens n darzustellen. Wenn auch Maple selbst vom Ergebnis überzeugt ist, wird entsprechend geantwortet.

```
> algsimplify := proc(x, n::posint)
  local mp, j, tst;
      Digits := 10*n;
      mp := PolynomialTools:-MinimalPolynomial(x, n);
      for j to degree(mp) do
          tst := convert(RootOf(mp, index=j), radical);
          evalf(x)=evalf(tst);
          simplify(expand(x-tst));
          if %=0 then return tst end if
      end do;
      return x
  end proc
```

Noch besser geht's so (das klappt auch später oft):

```
> N:=map( simplify@algsimplify, M, 6 ):
```

Stimmt das wirklich??

```
> simplify(M-N), evalf(M,50)-evalf(N,50);
```

Ja!

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0. & 0. & 0. \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{bmatrix}$$

B0 vereinfachen. Probe.

```
> oldB0:=B0:
  B0:=map(algsimplify,oldB0,6):
  evalf(B0-oldB0);
```

$$\begin{bmatrix} -0.40\,10^{-8} & 0.1\,10^{-9} & 0. \\ -0.31\,10^{-8} & 0. & 0. \\ 0. & 0. & 0. \end{bmatrix}$$

B1 vereinfachen. Probe.

```
> oldB1:=B1:
  B1:=map(algsimplify,oldB1,6):
  evalf(B1-oldB1);
```

$$\begin{bmatrix} 0. & 0.21\,10^{-8} & 0. \\ 0. & 0. & 0. \\ -0.1\,10^{-9} & 0.15\,10^{-8} & 0. \end{bmatrix}$$

```
> N,evalf(N);
```

$$\begin{bmatrix} 2^{(2/3)} & 0 & -\sqrt{2^{(2/3)}-1} \\ \sqrt{4-2\,2^{(1/3)}} & -2^{(1/3)} & -2^{(1/3)} \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 1.587401052 & 0. & -0.7664209366 \\ 1.216617401 & -1.259921050 & -1.259921050 \\ 0. & 0. & -1. \end{bmatrix}$$

```
latex(N);
```

Das Gittervolumen (die Determinante der Gitterbasis) sollte das Volumen unserer Körper sein, also 2!

```
> Determinant(N)=2;
```

$$2 = 2$$

Die Skalarprodukte der Gitterbasisvektoren:

```
> map( simplify@algsimplify, Transpose(N) . N, 6 );
  evalf( % );
```

$$\begin{bmatrix} 4 & -2\sqrt{2^{(2/3)}-1} & -\sqrt{3}\,2^{(2/3)} \\ -2\sqrt{2^{(2/3)}-1} & 2^{(2/3)} & 2^{(2/3)} \\ -\sqrt{3}\,2^{(2/3)} & 2^{(2/3)} & 2\,2^{(2/3)} \end{bmatrix}$$

$$\begin{bmatrix} 4. & -1.532841873 & -2.749459275 \\ -1.532841873 & 1.587401052 & 1.587401052 \\ -2.749459275 & 1.587401052 & 3.174802104 \end{bmatrix}$$

Die Gitterbasis, wie wir sie ab jetzt verwenden werden:

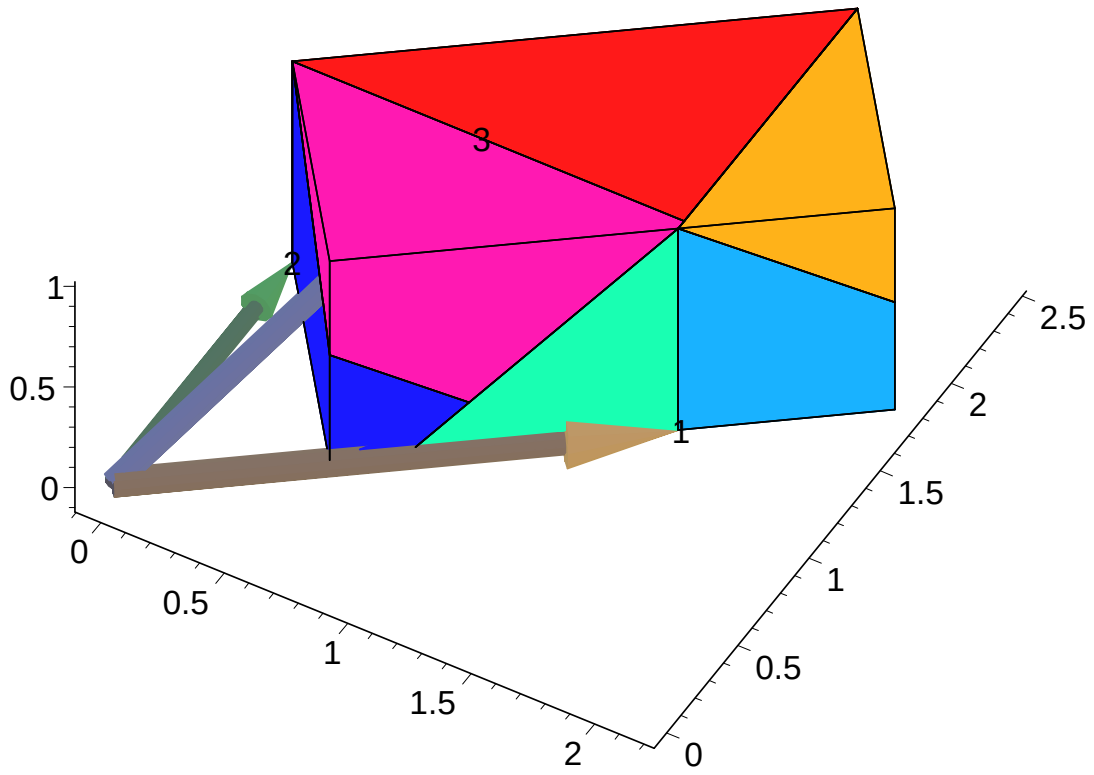
```
> G:=N.<<1,0,0>|<0,-1,0>|<0,0,-1>>;
```

$$G := \begin{bmatrix} 2^{(2/3)} & 0 & \sqrt{2^{(2/3)}-1} \\ \sqrt{4-2\,2^{(1/3)}} & 2^{(1/3)} & 2^{(1/3)} \\ 0 & 0 & 1 \end{bmatrix}$$

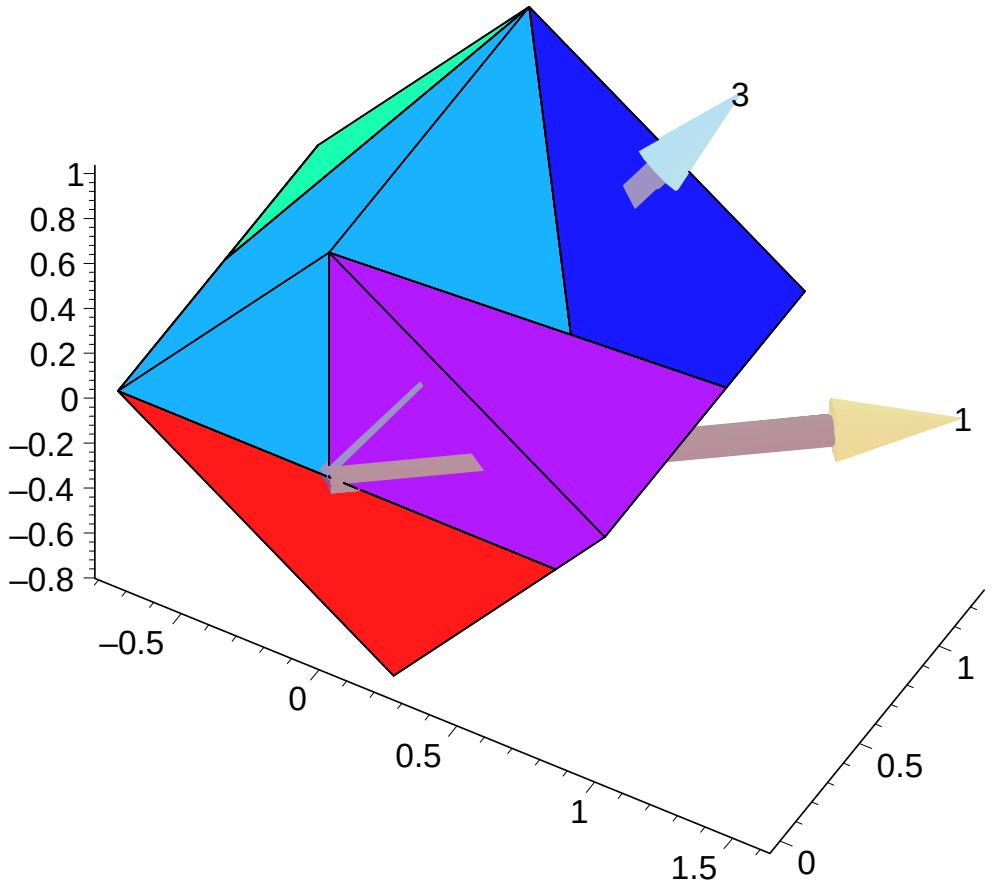
Alle Bausteine nochmal zeichnen für eine Zwischenkontrolle:

```
> with(plots,arrow):
  Gitterbasis:=arrow({seq(G[1..-1, i], i=1..3)},
  scaling=CONSTRAINED, axes=FRAMED),
  map(textplot3d,[seq([op(convert(G[1..-1,i],list)),convert(i,s
  tring)],i=1..3)],color=black):
  Altar:=eval( cut_cube7_fill(0,-1,-1), TEXT=(()->NULL) ):
  Wurfel:=eval( cut_cube7_fill(-1,0,-1), TEXT=(()->NULL) ):
  Gitterpunkte:=eval( cut_cube7_fill(-1,-1,1), TEXT=(()->NULL)
  ):
```

```
> display(Gitterbasis,Altar,scaling=constrained);
```



```
> display(Gitterbasis,Wurfel,scaling=constrained);
```



In das Gitter lassen sich verschiedene Quader einpassen. Dazu muss man nur eine Gitterbasis nach Gram-Schmidt orthogonalisieren und dann mit 0 bis 1 skalieren. Diese Funktion bestimmt die aufspannenden Vektoren.

```
> findcuboid:=proc(G::Matrix)
    GramSchmidt( [seq(G[1..-1,i],i=1..ColumnDimension(G))] );
    map( x->map(algsimplify,x,12), % );
    %, map( x->algsimplify(Norm(x,2),12), % );
  end proc:
> G0,G0n:=findcuboid(G,6):
  G0n;
```

$$[2, 1, 1]$$

```
> G1,G1n:=findcuboid( G . <<0,1,0>|<0,0,1>|<1,0,0>>, 6 ):
  G1n;
```

$$[2^{(1/3)}, 2^{(1/3)}, 2^{(1/3)}]$$

```
> G2,G2n:=findcuboid( G. <<0,0,1>|<1,0,0>|<0,1,0>>, 6 ):
  G2n;
```

$$\left[32^{(1/6)}, \sqrt{-\frac{3\,2^{(2/3)}}{2}+4}, \frac{\sqrt{2424+3232\,2^{(1/3)}+909\,2^{(2/3)}}}{101}\right]$$

```
> mysimplify(convert(G2n,'*'));
```

$$2$$

```
> G3,G3n:=findcuboid( G. <<1,0,0>|<0,0,1>|<0,1,0>>, 6 ):
  G3n;
```

$$\left[2, \frac{2^{(2/3)}\sqrt{-3+4\,2^{(1/3)}}}{2}, \frac{\sqrt{2424+3232\,2^{(1/3)}+909\,2^{(2/3)}}}{101}\right]$$

```
> G4,G4n:=findcuboid( G. <<0,0,1>|<0,1,0>|<1,0,0>>, 6 ):
  G4n;
```

$$\left[2^{(5/6)}, \frac{2^{(5/6)}}{2}, 2^{(1/3)}\right]$$

```
> G5,G5n:=findcuboid( G. <<0,1,0>|<1,0,0>|<0,0,1>>, 6 ):
  G5n;
```

$$[2^{(1/3)}, 2^{(2/3)}, 1]$$

Es gibt noch viele andere Möglichkeiten ...

```
> <<2,3,0>|<1,2,0>|<0,0,1>>:
  Determinant(%);
  G6,G6n:=findcuboid( G.%% , 3):
```

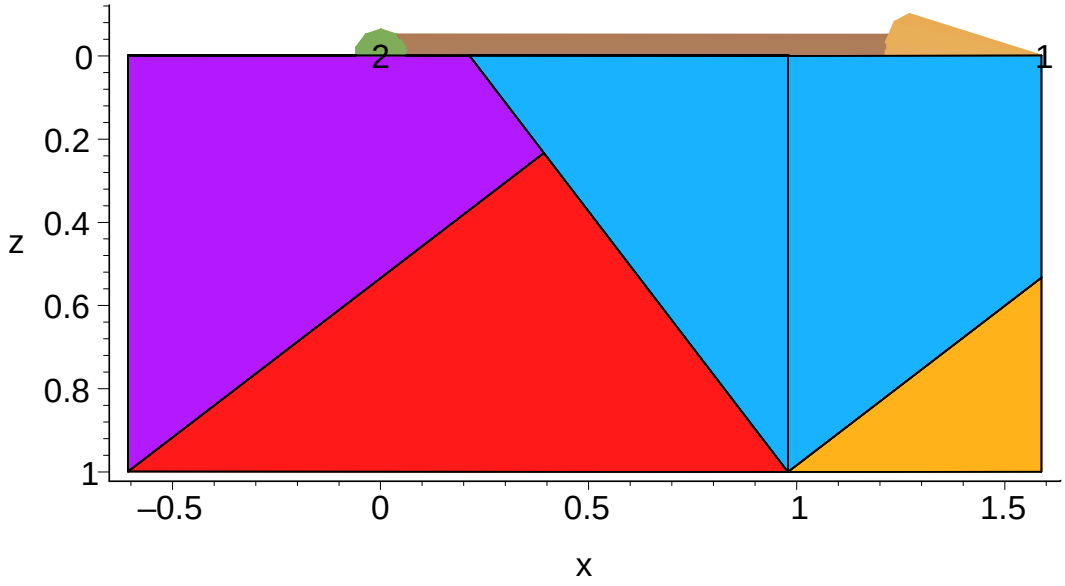
$$1$$

```
> evalf(G6n);
```

$$[6.977156437, 0.2866497288, 1.]$$

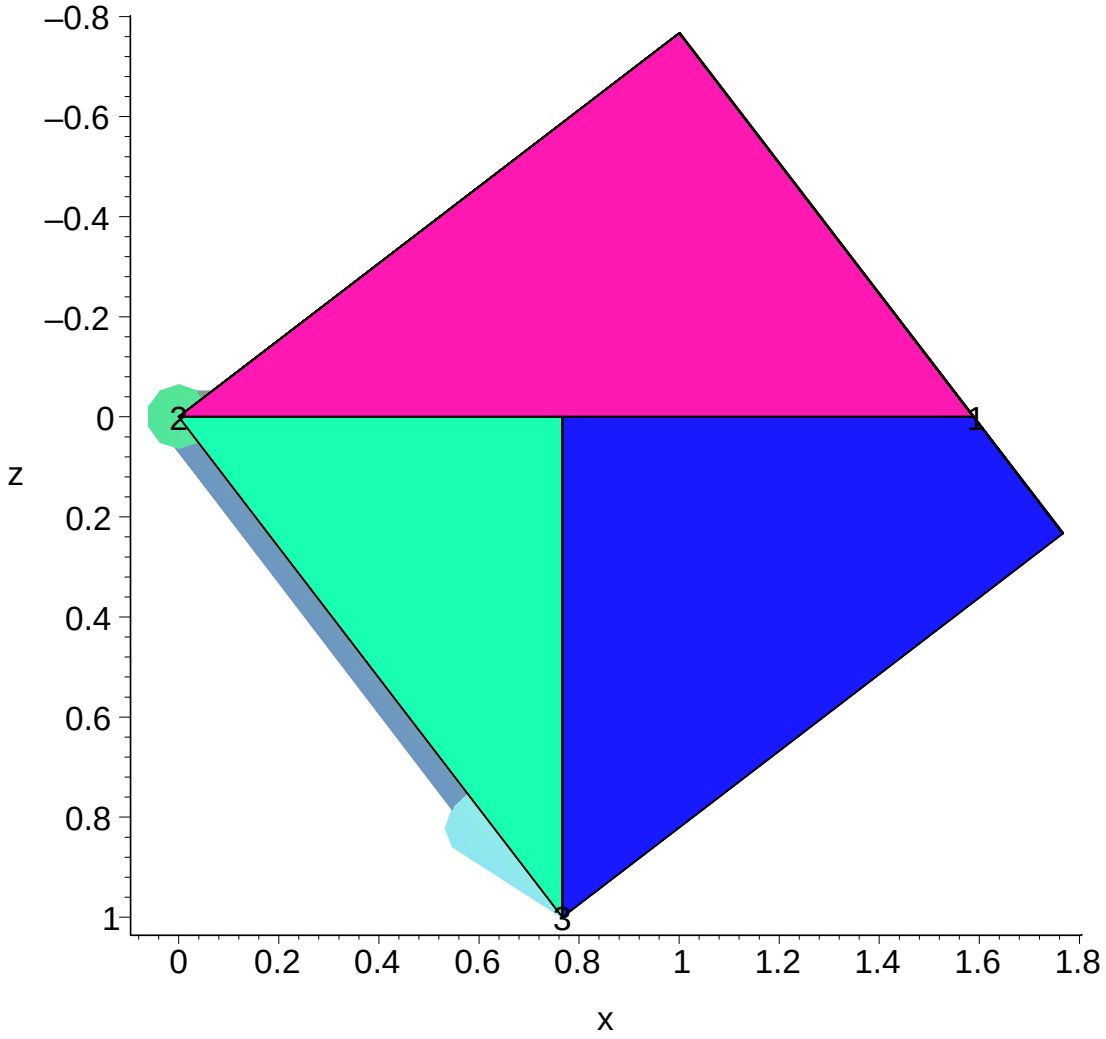
Die Verschiebung des Altars.

```
> S0:=Matrix(G1).<1,0,0>-Matrix(G0).<0,1,0>:
  display(Gitterbasis,translate(Altar,op(convert(-S0,list))),
  scaling=constrained,labels=["x","y","z"],
  orientation=[-90,-90]);
```



Die Verschiebung des Würfels.

```
> S1:=Matrix(G0).<0,0,1>-Matrix(G1).<0,1,0>:
  display(Gitterbasis,translate(Wurfel,op(convert(-S1,list))),
  scaling=constrained,labels=["x","y","z"],
  orientation=[-90,-90]);
```




```
> cG0:=transform((a,b,c)->convert(Matrix(G0).<a,b,c>,list))
  (cuboid( [0,0,0], [1,1,1] )):
  cG1:=transform((a,b,c)->convert(Matrix(G1).<a,b,c>,list))
  (cuboid( [0,0,0], [1,1,1] )):
  cG5:=transform((a,b,c)->convert(Matrix(G5).<a,b,c>,list))
  (cuboid( [0,0,0], [1,1,1] )):
```

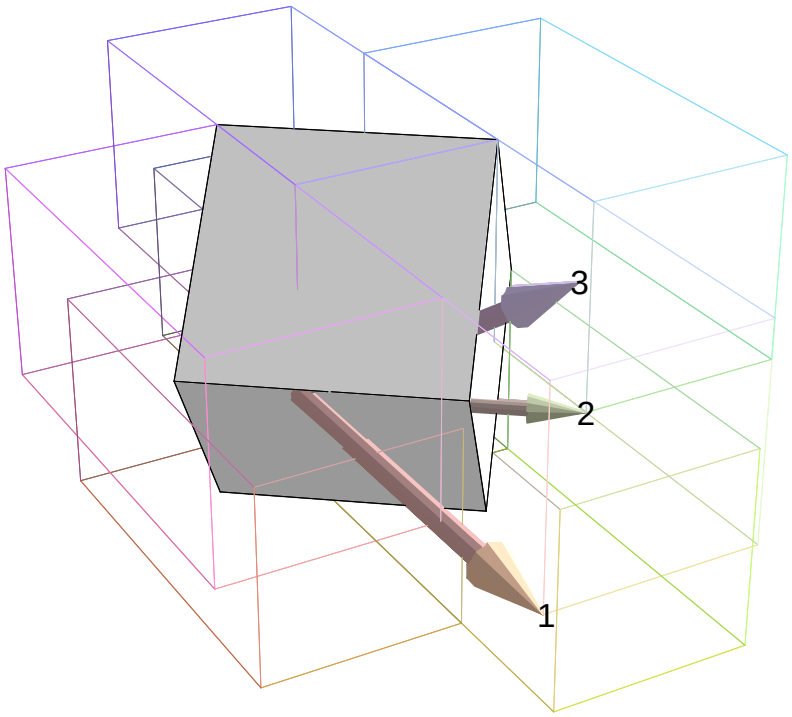
Die Verschiebung zwischen Altar und Würfel. (Die Ecke [0,1,1] des Altars und die Ecke [1,1,0] des Würfels sind am selben Ort.)

```
> S:=map( algsimplify,
          Matrix(G0).<0,1,1>-Matrix(G1).<1,1,0>,
          6 );
```

$$S := \begin{bmatrix} -\frac{27^{(1/6)}\,4^{(5/6)}}{4} \\ -2^{(1/3)} + \frac{2^{(2/3)}}{2} \\ 0 \end{bmatrix}$$

```
> cs:=NULL:
  for t in [
   [-1, 1, 0],
   [-1, 1,-1],
   [-1, 0, 0],
   [ 0, 1,-1],
   [ 0, 0, 0],
   [ 0, 0,-1],
   [ 0,-1, 0]
  ] do
    G . <op(t)>:
    cs:=cs,translate( cG0, op(convert(%,list))):
  end do:
  Doppelkachelung:=display(
    Gitterbasis,
    cs,

  translate(display(cG1,style=patch,color=white),op(convert(S,l
  ist))),
    scaling=constrained, labels=["x","y","z"],axes=none ):
  #Doppelkachelung;
> display( Doppelkachelung,
    style=wireframe,thickness=1,
    orientation=[8,63],projection=0.5,lightmodel=light4);
```



[>

Maple-Bilder in pstricks verwandeln für den Artikel

[>