

Bibliotheken und Übersetzen

Marco Feuerstein

16. Mai 2006

chair for computer aided medical procedures

department of computer science | technische universität münchen

Was passiert genau, wenn man den Compiler aufruft?

- Compiler wird meist synonym für die Sammlung von Programmen [Compiler Collection] verwendet, die man zur Programmerstellung benötigt
- Folgende Teile werden beim Aufruf vom gcc, cc, mcc, ... aufgerufen:
 - Präprozessor
Textersetzungen
 - Übersetzer [Compiler]
Übersetzt Programmfragmente in C++ zu Programmfragmenten in Binärcode
 - Verknüpfer [Linker]
Verknüpft Binärprogrammteile zu einem Programm

Präprozessor

- Macht Textersetzungen *vor* dem Übersetzen
- hilft, verschiedene Versionen vom Programm zu erstellen (Debug vs. Release, Un*x vs. Win32, etc.)
- alle Anweisungen für den Präprozessor fangen mit # an
 - `#include "file.h"` fügt die Datei file.h ein
 - `#define BLUBB 1` ersetzt überall im Programmtext BLUBB mit 1
 - `#ifdef BLUBB`
Code1
#else
Code2
#endif
fügt im Programmtext Code1 ein falls BLUBB definiert wurde,
ansonsten Code2

Präprozessor

- Schutz davor, dass man den Header mehrmals aus Versehen aufruft:
- Variablen (eigentlich Ersetzungsworte) zur Übersichtlichkeit immer mit Großbuchstaben benennen!
- Wenn es geht, so wenig wie möglich mit dem Präprozessor arbeiten!

```
#ifndef MEINEKLASSE_H
#define MEINEKLASSE_H
#include ...
class MeineKlasse {...};
#endif
```

MeineKlasse.h

Bibliotheken [Libraries]

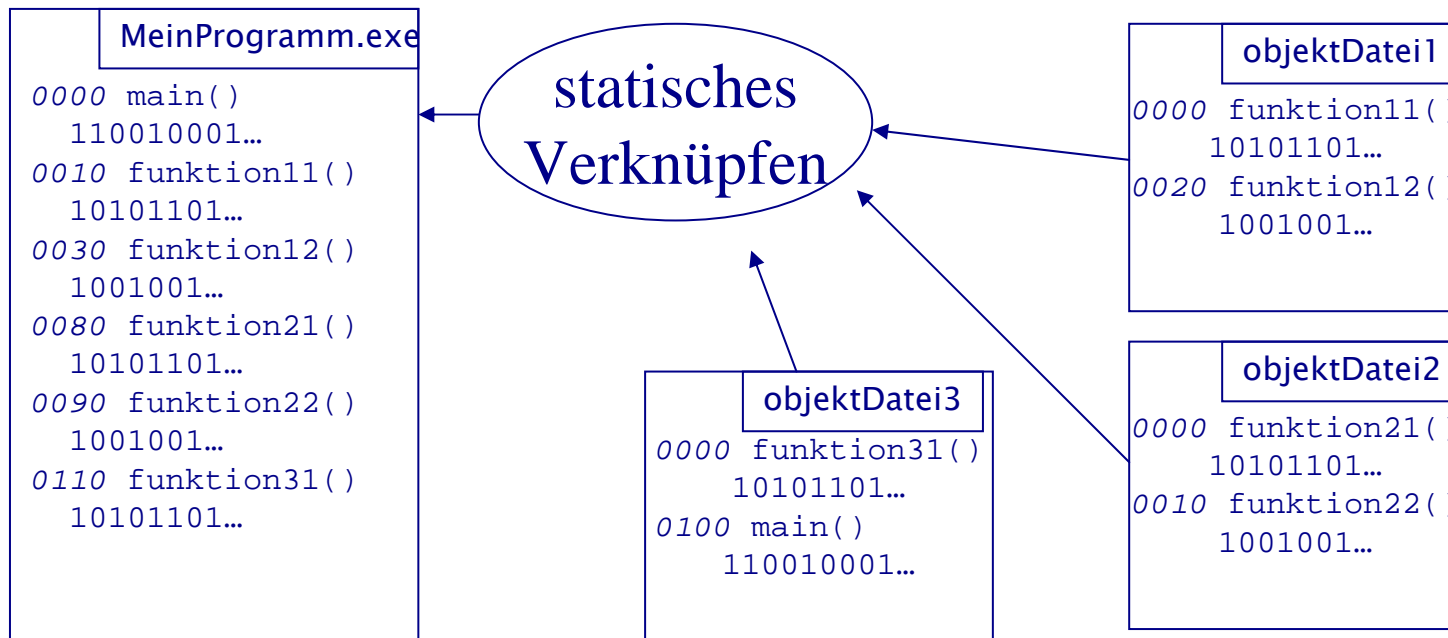
- Bibliotheken (*.lib) sind vorkompilierte Binärdateien ohne Einstiegspunkt [Entrypoint]
 - damit man Programmteile, die sehr oft benutzt werden, nicht immer wieder neu kompiliert werden müssen (Zeitvorteil)
 - damit man Programmteile weitergeben kann, ohne den Programmtext zeigen zu müssen (Rechtlicher Grund)
 - damit man Programmteile weitergeben kann und sich sicher ist, dass er nicht geändert wurde (Programmiertechnischer Vorteil)
 - damit man beliebte Programmteile nur einmal auf der Festplatte (oder im Speicher) vorrätig haben muss (Platz- und Geschwindigkeitsvorteil)

Einstiegspunkt?!

- Der Einstiegspunkt [Entrypoint] eines Programms ist der Teil, an dem das Programm starten soll. Unter C++ ist es die globale Funktion `int main()`, die es selbstverständlich nur einmal geben darf
- Ein Programm ohne Einstiegspunkt ist allein nicht lauffähig

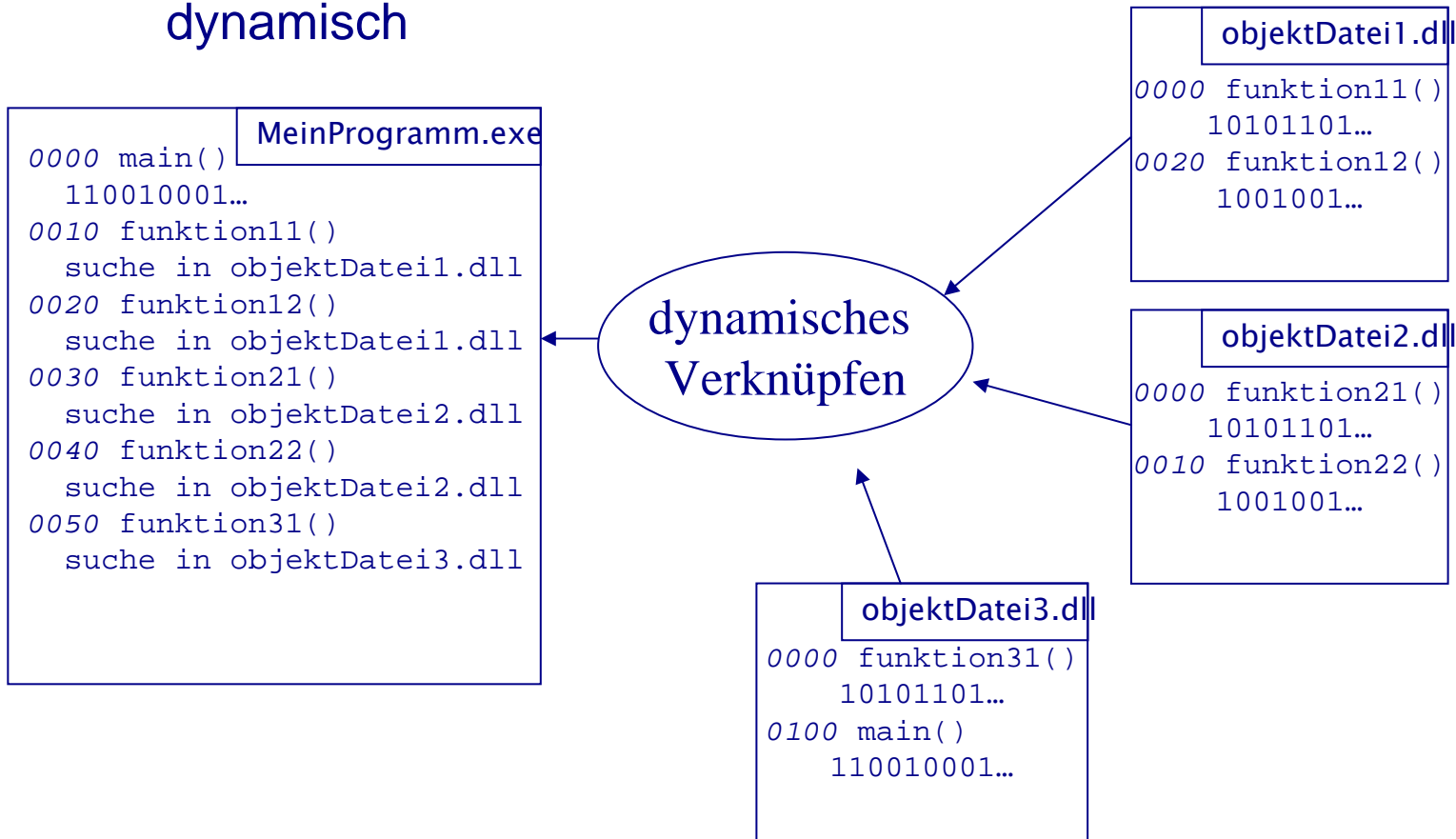
Verknüpfen (link)

- Das eigentliche Programmerstellen übernimmt der Linker
- Er kopiert den Code aus den Bibliotheken und ersetzt im Code die symbolischen Funktionen mit konkreten Adressen



Verknüpfen (link)

- Die zweite Art der Verknüpfung funktioniert zur Laufzeit, d.h. dynamisch



Gegenüberstellung

statisches Verknüpfen

- einfachste (auch älteste) Methode
- ein große Datei
- portable Verknüpfungsmethode
- neue Version der Bibliothek → ganze Datei muss neu generiert werden

dynamisches Verknüpfen

- komplizierter Mechanismus, der auf unterschiedlichen Betriebssystemen unterschiedlich angesprochen wird
- Realisierung von Plugins möglich
- Weniger Speicherbedarf
- Man kann eventuell falsche (Version einer) Bibliothek mitliefern
- neue Version der Bibliothek → nur Bibliothek muss neu generiert werden

Tipps zu Compiler-/Linkerfehlern

- Wenn bei der Verknüpfung von Programmen Fehler auftreten, bedeutet es meist, dass in den angegebenen Objektdateien ein Symbol (Funktion, Variable, Klasse, ...) nicht gefunden werden konnte
- Der Grund dafür ist in den meisten Fällen
 - ein Schreibfehler
 - falscher oder fehlender Namensraum
 - `virtual / static` vergessen
- Hinweis 1: Der Compiler prüft *nie* nach, ob eine deklarierte Funktion (im Header `xx.h`) tatsächlich (in der Implementierung `xx.cpp`) implementiert wurde
- Hinweis 2: Der Compiler beschwert sich immer, wenn eine nicht deklarierte Funktion implementiert wird
- Hinweis 3: Manchmal hilft es, *alles* (inklusive Bibliothek) neu zu bauen

Wenn ich fremde Bibliotheken benutzen möchte, muss ich

- im Programmtext den Header einbinden, in dem die gewünschten Objekte oder Funktionen bereitgestellt werden
- dem Linker den/die Dateinamen der Bibliotheken mitteilen, damit diese in das Programm verknüpft werden können
- in der IDE oder im makefile den Pfad des Headers **und** der Bibliothek angeben

Erste Schritte in OpenGL

Marco Feuerstein

16. Mai 2006

chair for computer aided medical procedures
department of computer science | technische universität münchen

Open GL (Graphics Library) seit 1992

- API Spezifikation für Plattform-, Programmiersprache-, Hardware-unabhängigen Zugriff auf (beschleunigte) Graphikhardware
- Dessen Implementierung wird i.A. mit dem Graphikkartentreiber mitgeliefert
- Zustandsbasiertes Modell

Mit OpenGL kann man nicht nur einfach,
sondern auch schnelle Graphik erzeugen

Was ist zu tun?

- GLUT Bibliothek einbinden (glut32.lib aus dem GLUT Verzeichnis)
 - GLUT ermöglicht, ein Fenster mit GL Kontext unter verschiedenen Betriebssystemen zu erzeugen
 - GLUT ermöglicht, Tastatur, Maus und Keyboardeingaben zu behandeln
 - GL Bibliothek einbinden (glu32.lib, opengl32.lib) – das Verzeichnis ist vom Betriebssystem her bekannt
 - Im Header des Programms glut.h und glu.h mit `#include` einfügen
- Nun steht alle Funktionalität zur Verfügung!

Was ist zu tun?

- Fenster mit GL Kontext erzeugen

```
glutInit();  
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE); // Display Modus (Farbe und 2 Puffer)  
glutInitWindowSize(500, 500); // Anfangsgröße des Fensters  
glutCreateWindow(„Programmierpraktikum SS 2006“); // Fenstertitel
```

- Renderfunktion und Tastaturfunktion registrieren

```
glutDisplayFunc(render);  
glutKeyboardFunc(keyboard);
```

- Hauptschleife aufrufen

```
glutMainLoop();
```

- Renderfunktion und Tastaturfunktion implementieren

```
static void render(void) {...};  
static void keyboard(unsigned char key, int x, int y)
```

Ein Dreieck

□ Puffer löschen

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Bildschirm und Tiefenpuffer löschen
glLoadIdentity(); // Modelview Matrix initialisieren
glTranslatef(-1.5f,0.0f,-6.0f); // 1,5 Einheiten nach links,
// 6 nach hinten
```

□ Dreieck zeichnen

```
glBegin(GL_TRIANGLES);
glVertex3f( 0.0f, 1.0f, 0.0f); // Oben
glVertex3f(-1.0f,-1.0f, 0.0f); // Unten links
glVertex3f( 1.0f,-1.0f, 0.0f); // Unten Rechts
glEnd(); // Ende des Dreiecks
```

□ Das Bild anzeigen

```
glutSwapBuffers();
```

□ Vierecke mit GL_QUAD, Polygone mit GL_POLYGON

□ Farbe mit

```
glColor3f(1.0f,0.0f,0.0f); // Farbe rot
```

Hilfe?!

- Delphi Wiki (<http://www.delphigl.com/>), ist zwar nicht C++, sondern Delphi, aber OpenGL ist sprachunabhängig!
- NeHe Online Tutorials samt Code (<http://nehe.gamedev.net/>) oder dessen deutsche Übersetzung (http://www.joachimrohde.com/cms/xoops/modules/articles/index.php?cat_id=1)
- Originalreferenz von SGI (<http://www.opengl.org/>)
- Das RedBook Version 1.0 online (http://www.opengl.org/documentation/red_book/)
- GLUT Spezifikation (<http://www.opengl.org/resources/libraries/glut/spec3/spec3.html>)

Koordinatensysteme & Transformationen

Marco Feuerstein

16. Mai 2006

Chair for Computer Aided Medical Procedures & Augmented Reality

Department of Computer Science | Technische Universität München

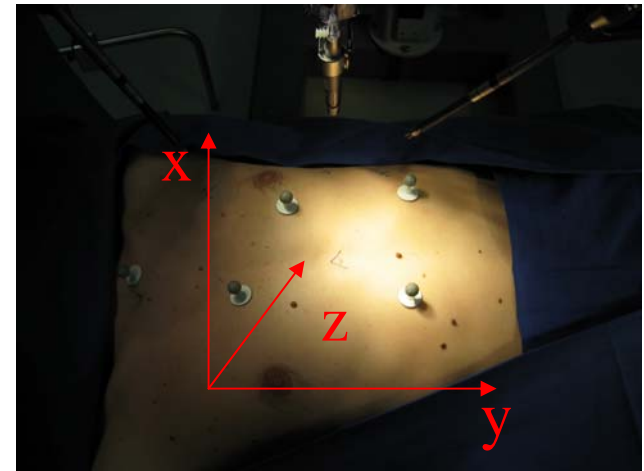
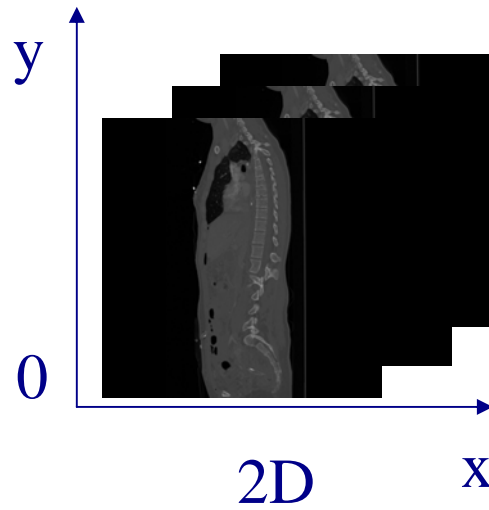
Inhalt

- Koordinatensysteme
- 1D Transformationen
 - Euklidisch
 - Metrisch
 - Affin
 - Projektiv
- 2D Transformationen
- 3D Transformationen -> nächste Stunde

Koordinatensysteme



1D

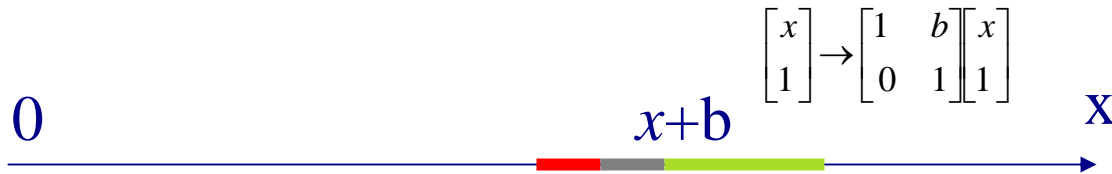


3D

1D Transformationen



1D

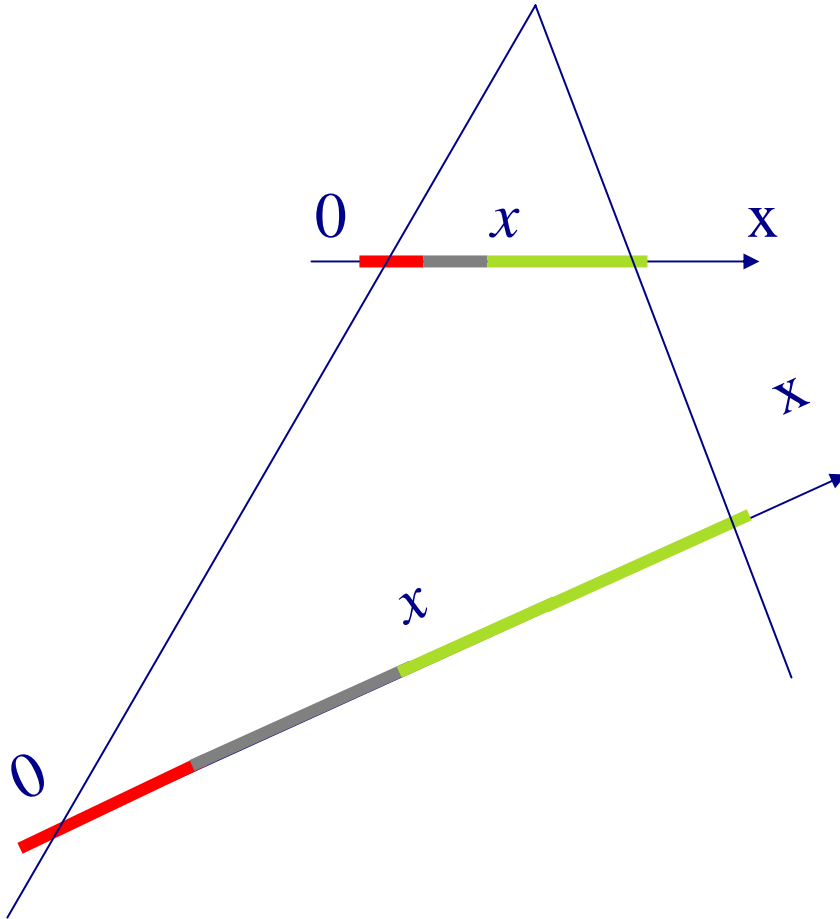


Euklidisch



Metrisch

1D Projective Transformation (P_1)

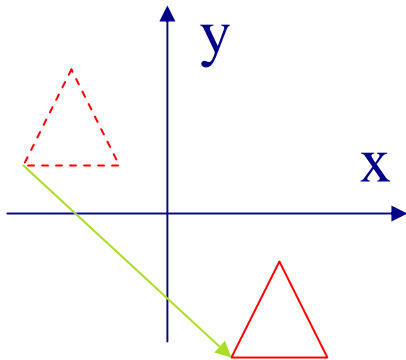


$$\begin{bmatrix} x \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} a & b \\ c & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}$$

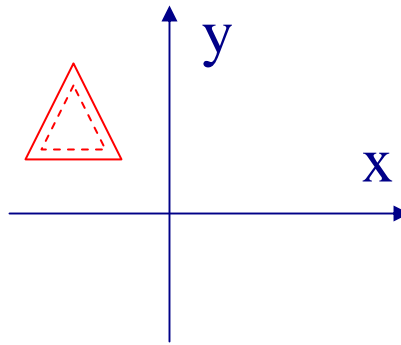
$$x \rightarrow \frac{ax + b}{cx + 1}$$

2D Transformationen

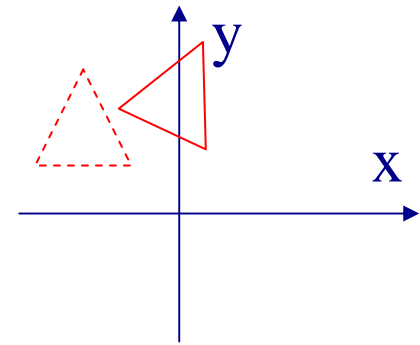
Einfache Koordinaten-Transformationen



Translation

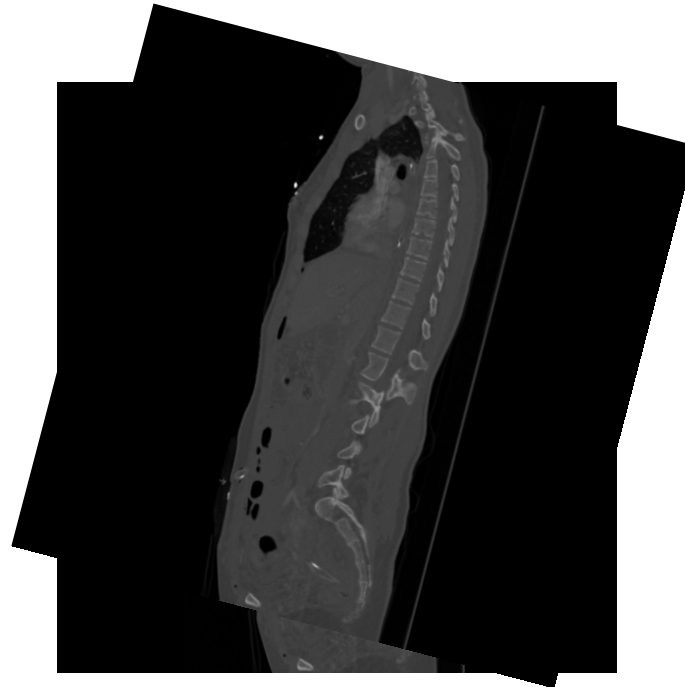


Skalierung



Rotation

2D Transformationen



Rotation

2D Rotation

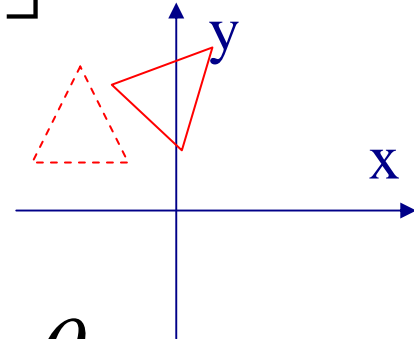
Rotation um Ursprung um Winkel θ

Rotationsmatrix

$$R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Transformierter Punkt

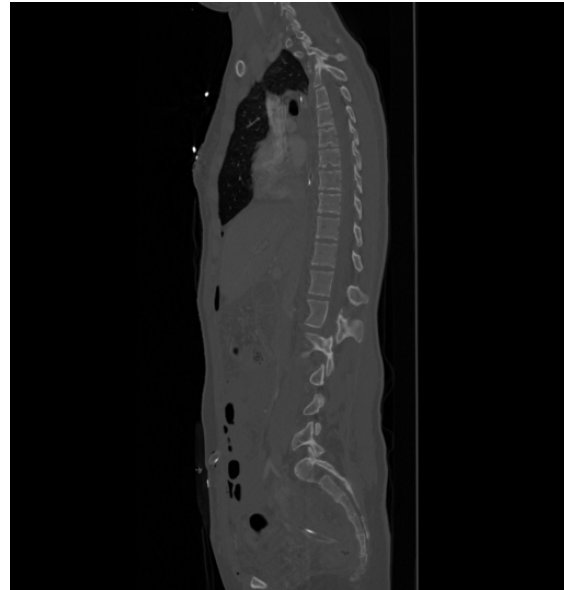
$$\vec{p}' = R\vec{p}$$



wobei $x' = x \cdot \cos \theta - y \cdot \sin \theta$

$$y' = x \cdot \sin \theta + y \cdot \cos \theta$$

2D Transformationen



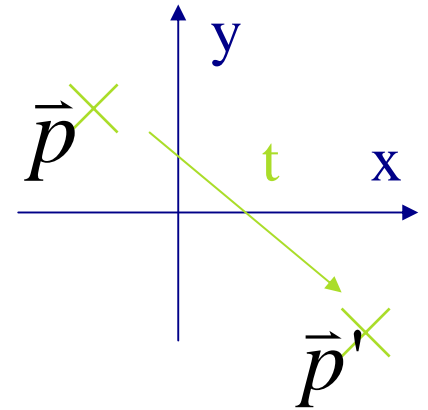
Translation

2D Transformationen

Originalpunkt $\vec{p} = \begin{bmatrix} x \\ y \end{bmatrix}$

Transformierter Punkt $\vec{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$

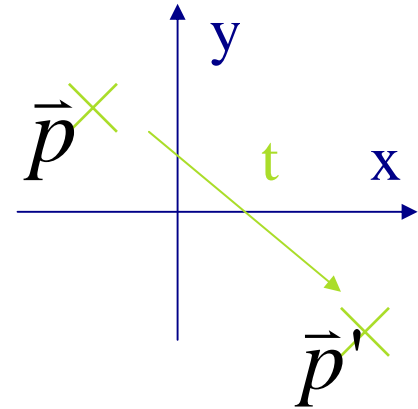
Translation $\vec{p}' = \vec{p} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$



2D Transformationen

Homogener Punkt

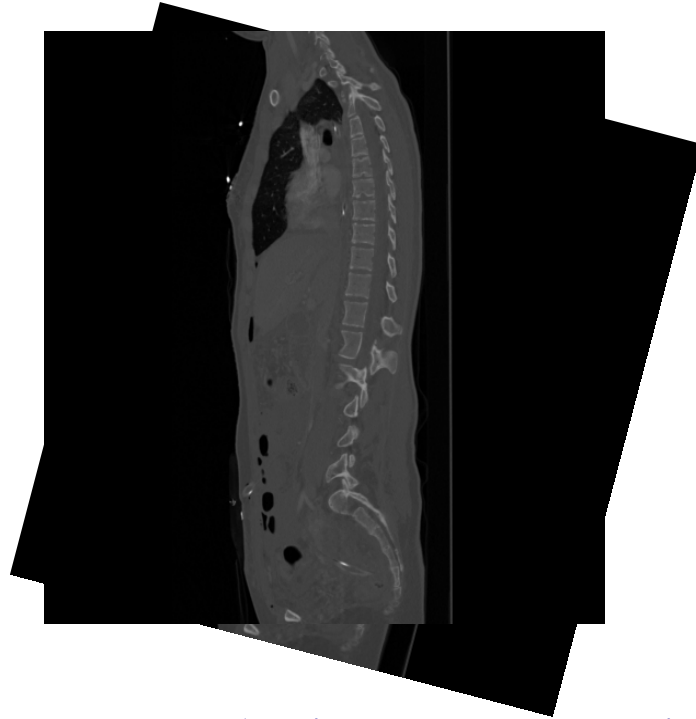
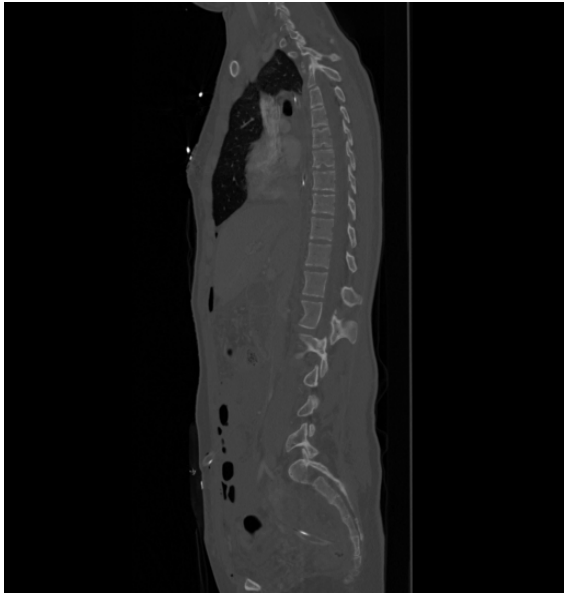
$$\vec{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Transformationsmatrix

$$\vec{p}' = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D Transformationen

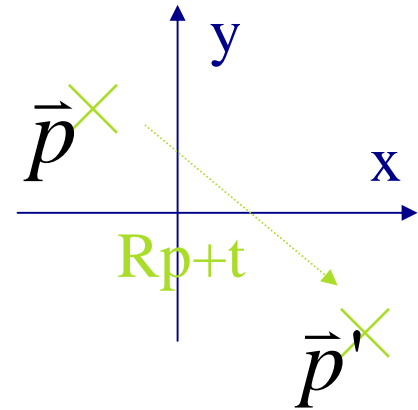


Euklidische Transformation (Translation & Rotation)

2D Transformationen

Homogener Punkt

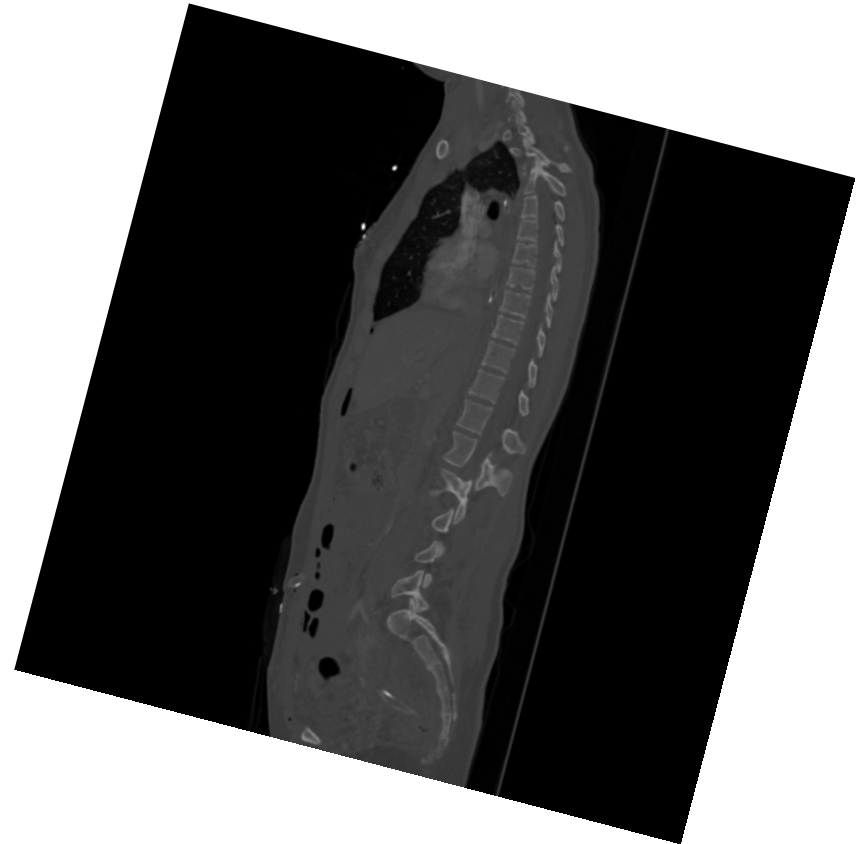
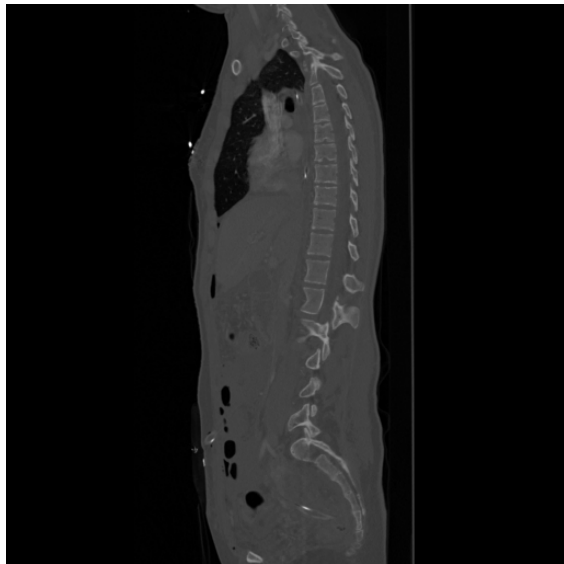
$$\vec{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Euklidische
Transformation

$$\vec{p}' = \begin{bmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D Transformationen



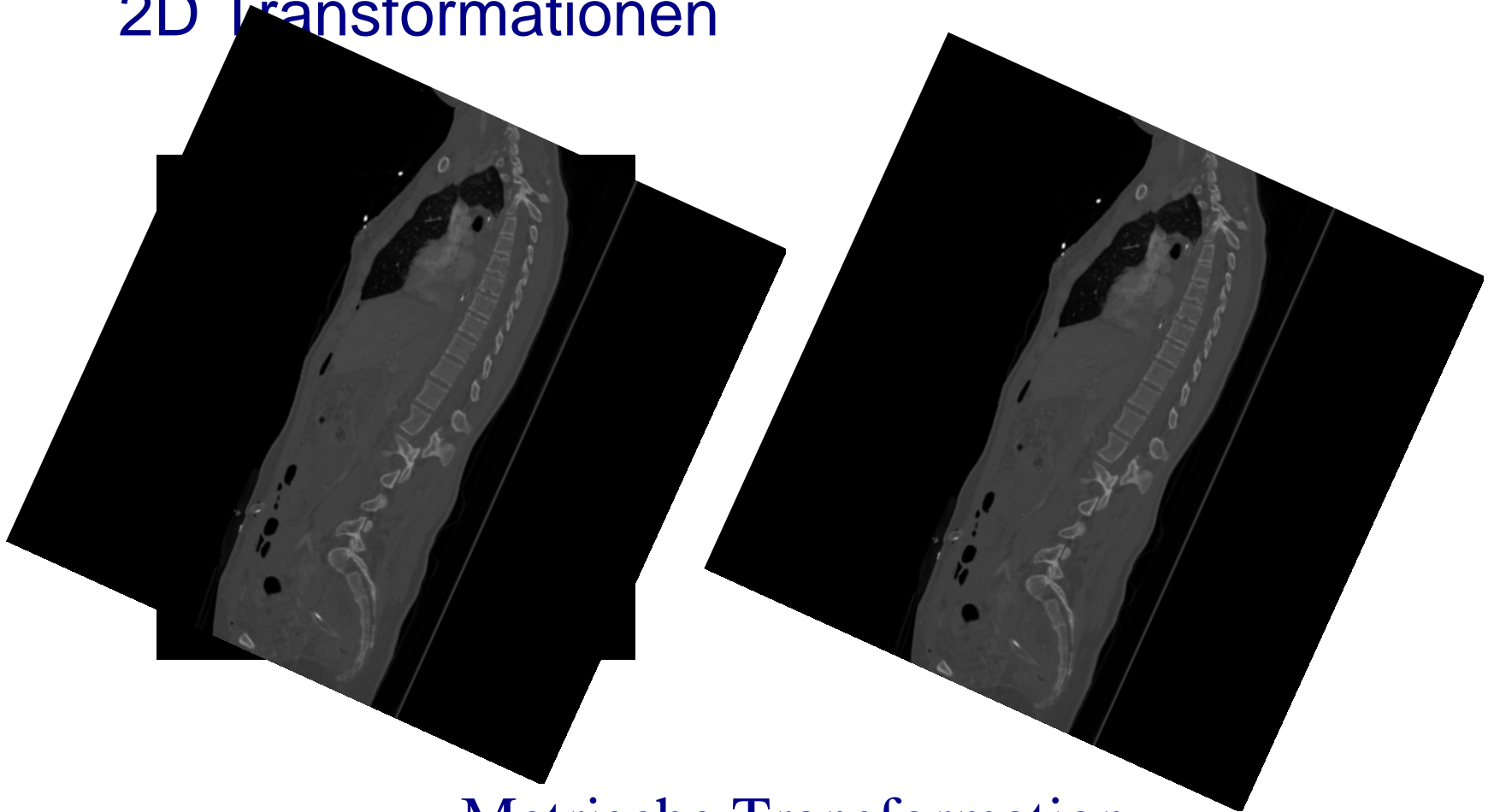
2D Transformationen

$$\begin{bmatrix} \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} & \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix} \end{bmatrix}$$

Metrische
Transformation

$$\vec{p}' = \begin{bmatrix} s \cdot \cos \theta & s \cdot -\sin \theta & t_x \\ s \cdot \sin \theta & s \cdot \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D Transformationen



Metrische Transformation

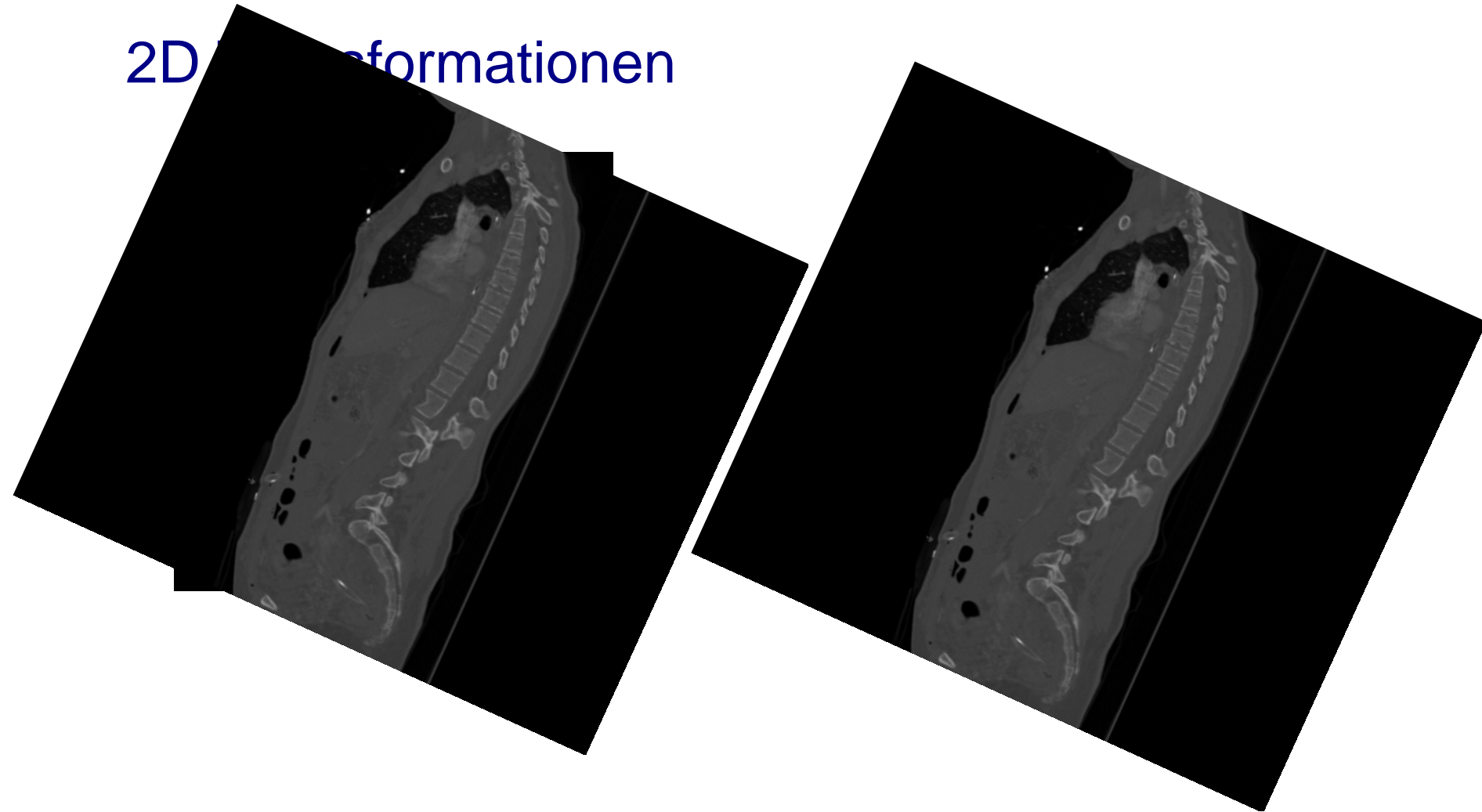
2D Transformationen

$$\begin{bmatrix} \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} & \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix} \end{bmatrix}$$

Metrische
Transformation

$$\vec{p}' = \begin{bmatrix} s_1 \cdot \cos \theta & s_1 \cdot -\sin \theta & t_x \\ s_2 \cdot \sin \theta & s_2 \cdot \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D Transformationen

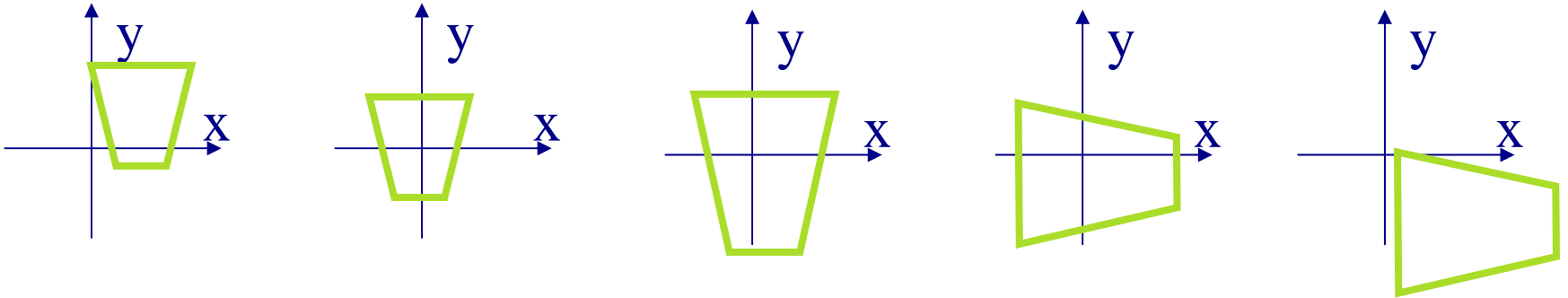


Nicht-isotropische Skalierung (Affine Transformation)

2D Transformation

Kombinationen von einfachen Transformation

$$\vec{p}' = T_2 + (R \cdot S \cdot (T_1 + \vec{p}))$$



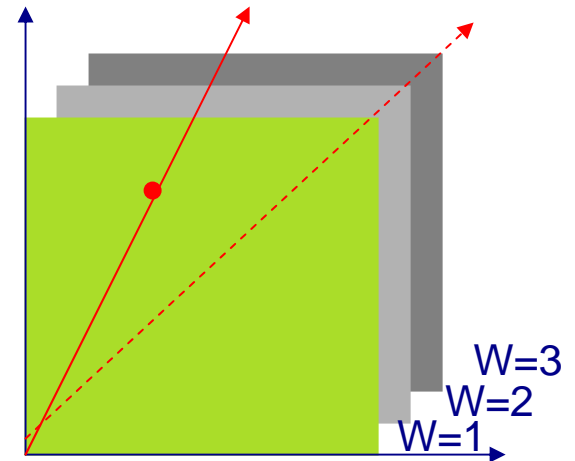
[J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes,
Computer Graphics: Principles and Practice, 2nd ed., Addison-Wesley, 1997]

Homogene Koordinaten

2D Punkt $\vec{p} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}$, wobei $w \neq 0$
und typischerweise $w = 1$

\vec{p}_i identisch mit \vec{p}_j , falls

$$\frac{1}{w_i} \cdot \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix} = \frac{1}{w_j} \cdot \begin{bmatrix} x_j \\ y_j \\ w_j \end{bmatrix}$$



2D Transformationen in OpenGL

Translation

```
glTranslatef(12.0f,0.0f,0.0f); // 12 Einheiten nach rechts verschieben
// (in positiver x-Richtung)
```

Rotation

```
glRotatef(30.0f,0.0f,0.0f,1.0f); // 30 Grad auf der x-y-Ebene rotieren
// (Gegen den Uhrzeigersinn)
```

Skalierung

```
glScalef(0.5f,0.5f,1.0f); // in x- und y-Richtung
// um Faktor 0.5 skalieren
```

Metrische Transformation in homogenen Koordinaten

```
glMultMatrixf(pointerToMatrix); // Multipliziert die aktuelle 4x4
// Matrix mit der 4x4 Matrix, auf die
// von pointerToMatrix gezeigt wird
```

Vorsicht: Bei Matrixmultiplikationen für Transformationen

- Muss man sich im GL_MODELVIEW mode befinden!
- Darf man oft ein glLoadIdentity() nicht vergessen!
- Darf man nicht vergessen, dass
 - OpenGL die neue Matrix von rechts multipliziert
 - OpenGL seine Matrizen im Speicher spaltenweise ablegt:

$$\begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}$$

Quellen

- Olivier Faugeras; *Three-Dimensional Computer Vision (Artificial Intelligence)*; 1993
- Richard Hartley, Andrew Zisserman; *Multiple View Geometry in Computer Vision*; 2003
- Emanuele Trucco, Alessandro Verri; *Introductory Techniques for 3-D Computer Vision*; 1998
- Gary Bishop, Greg Welch, and B. Danette Allen; *Course 11—Tracking: Beyond 15 Minutes of Thought*; 2001; <http://www.cs.unc.edu/~tracker/ref/s2001/tracker/>