DIGITALE AKUSTISCHE KARTOGRAPHIE

DIGITALE AKUSTISCHE KARTOGRAPHIE

Diplomarbeit

Daniel Rothaug

Fachhochschule Würzburg-Schweinfurt

Fachbereich Gestaltung

Kommunikationsdesign

9. Semester - Wintersemester 2004/2005

Prüfer:

Prof. Erich Schöls

Prof. Uli Braun

INHALT

„EINES TAGES WIRD DER MENSCH DEN LÄRM EBENSO
UNERBITTLICH BEKÄMPFEN MÜSSEN WIE DIE CHOLERA UND DIE PEST.“

Medizin-Nobelpreisträger Robert Koch (1843-1910)

AKUSTISCHE KARTOGRAPHIE

Das Ohr ist neben dem Auge unser wichtigstes Sinnesorgan. Mindestens die Hälfte unserer Wahrnehmung und Informationsgewinnung läuft über akustische Impulse. „Weghören“ ist nur im inhaltlichen oder sprichwörtlichen Sinn möglich. Auch unerwünschte akustische Einflüsse, z.B. Verkehrs-, Flug- oder Arbeitslärm, müssen gehört werden. Gesundheitliche Auswirkungen bei Dauerbelastung durch Lärm sind nur bedingt nachweisbar, werden aber inzwischen stark vermutet.

Einer Studie des Umweltbundesamtes (UBA) zufolge, wurde eine auffällige Verbindung von dauerbelastendem Straßenverkehrslärm an Verkehrskreuzungen und erhöhtem Herzinfarkt-Risiko bei Anwohnern festgestellt. Erst durch die ortsbezogene Abbildung (Kartographie) von Schall wurde dieser mögliche Zusammenhang sichtbar.

Geräusche und Lärm werden vorwiegend subjektiv empfunden und sind deshalb nur schwer oder durch Umschreibungen an andere vermittelbar. Im Gegensatz zur optischen Wahrnehmung ist es uns nur eingeschränkt möglich, akustische Signale zu archivieren, zu vergleichen und damit zu interpretieren. Wir sind auf die Abbildung von Akustik angewiesen.

„WER SCHNELLEN UND BLEIBENDEN EINDRUCK MACHEN WILL, BEDIENT SICH DER BILDER.“

Otto Neurath, 1933

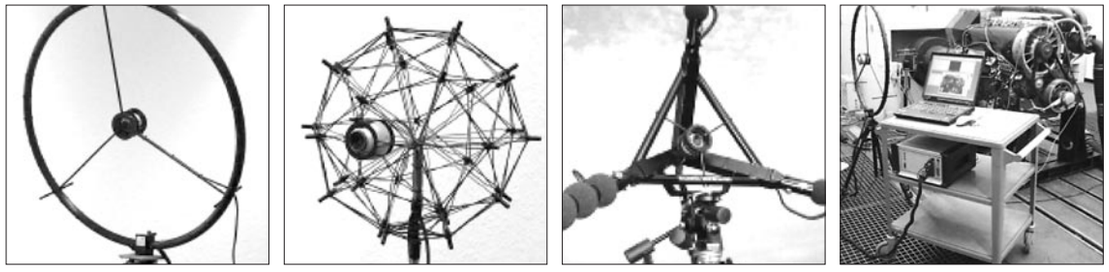
Vor allem in Forschung und Technik haben sich zahlreiche Abbildungsmodelle aus der Mathematik und Physik etabliert. Diese Visualisierungen dienen vorwiegend der wissenschaftlichen Auswertung von Zahlen und können nur mit entsprechendem Fach- und Hintergrundwissen interpretiert werden.

Physikalische Einheiten, wie der Schalldruckpegel in Dezibel (dB) und Frequenzen in Hertz (Hz), liefern uns zwar eine objektive Grundlage und ermöglichen den Vergleich akustischer Ereignisse, dienen aber in erster Linie nur als mathematische Denkwerkzeuge. So ist Dezibel z.B. eine logarithmische Einheit und gibt das Verhältnis zweier Werte nicht direkt wieder, sondern logarithmiert. Durch Bewertungsfilter (dBA, dBB, dBC oder dBD) wird versucht die Messung dem Frequenzempfinden unseres Gehörs anzupassen.

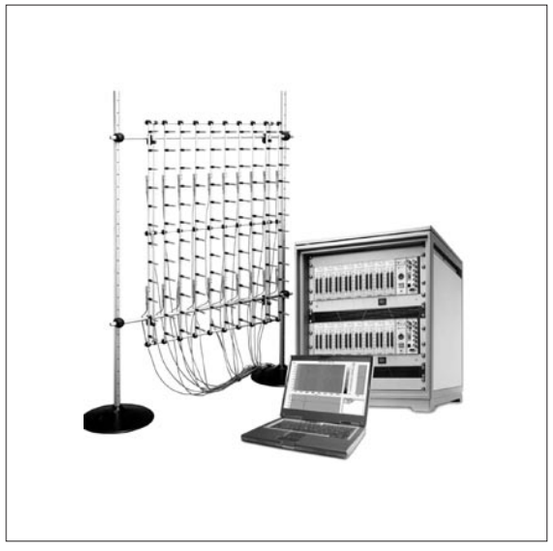
Kognitiv versuchen wir aber automatisch, akustische Eigenschaften mit uns vertrauten Metaphern gleichzusetzen. Das häufig bei Stereoanlagen oder Fernsehgeräten verwendete Lautstärke-Symbol impliziert, dass wir uns etwas Lautes größer und etwas Leises kleiner vorstellen.

Digitale Systeme bieten sich heute mehr denn je als Sensoren zur Erweiterung unserer eigenen Wahrnehmungsfähigkeit (z.B. in der Akustik) an. Sie erlauben uns (fast) unbegrenzt komplexe Daten zu erfassen, zu verarbeiten, zu archivieren und diese miteinander zu vernetzen.

Da uns aber grundsätzlich ein binäres Verständnis fehlt, sind wir bei der Kommunikation mit digitalen Endgeräten immer auf Informationsvisualisierungen angewiesen, die uns erlauben, diese Daten zu verstehen. Trotzdem greifen wir nach wie vor auf ursprünglich analoge Abbildungsformen zurück, die zur Veranschaulichung einfacher Datenarchitekturen konzipiert wurden.



Akustische Kameras für Flächen-, Raum- und Distanzmessung der Gesellschaft für angewandte Informatik, www.acoustic-camera.com



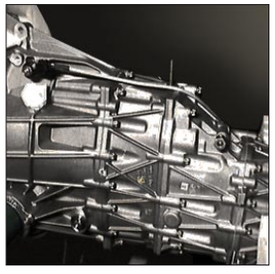
Non-Stationäres holographisches Messsystem von Brüehl&Kjær, Dänemark, www.bkhome.com

DATENERFASSUNG

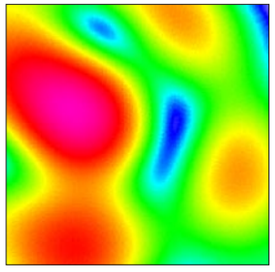
Vorraussetzung für die Kartographie von Schall ist die Erfassung aller akustischer Daten und des lokalisierten Referenzpunktes.

Die „Akustische Kamera“ und verschiedene Systeme der akustischen Holographie sind in der Industrie zur Fehleranalyse und Prototypen-Entwicklung bereits im Einsatz. Technische Grundlage dieser Methoden ist ein Matrix-Aufbau mit mehreren Mikrofonen (Array), in Kombination mit einer optischen Kamera. Durch die Berechnung des Laufzeitunterschiedes kann der Schall mit dem Mikrofon-Array lokalisiert und dem optischen Referenzbild zugeordnet werden. Diese Systeme sind - zumindest theoretisch - auch für den Einsatz von Echtzeitmessungen denkbar.

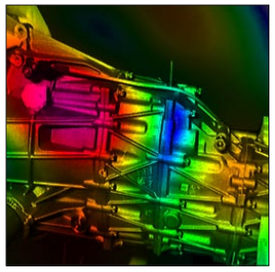
Aufgrund der Herstellungs- und Materialkosten sind diese Geräte nur schwer zugänglich und mit hohen Anschaffungskosten verbunden. In der Bau- und Schwingungstechnik wird neben umfangreichen Berechnungsverfahren auch auf Lasermessungen zurückgegriffen.



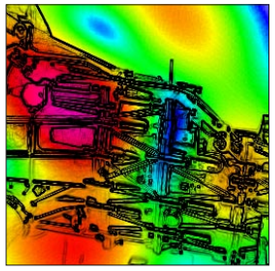
Originalbild eines Getriebes bei 1600 Hz in einem Bereich von 58 - 78 dBA.



Reine akustische Abbildung. Zwischen Originalbild und Schallbild besteht nur sehr wenig optischer Zusammenhang.



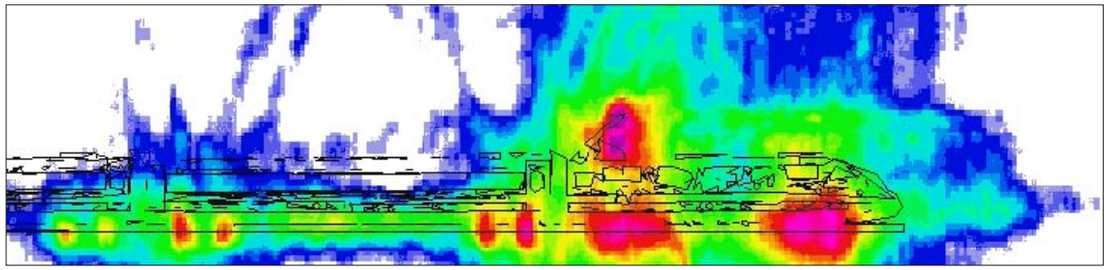
Multiplikation der ersten beiden Bilder. Photorealistische Details bleiben sichtbar, jedoch auf Kosten der akustischen Information.



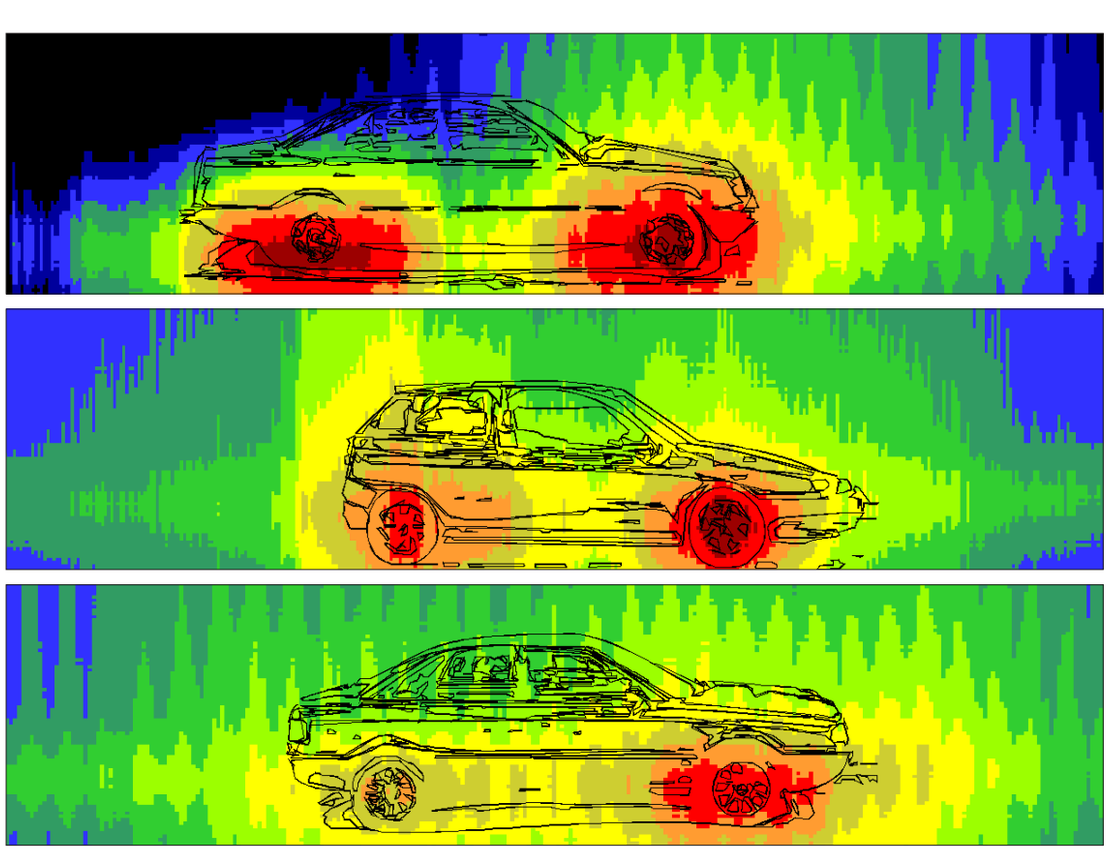
Kontur-Abbildung des Originalbildes. Es bleiben nur wenig photographische Details sichtbar, der Schall wird besser erkennbar.

Allen Messverfahren ist eine ähnliche Abbildung der Daten durch Spektralbilder gemeinsam. Obwohl diese Methode eine analoge, physikalische Herkunft hat, wird sie auch heute noch für digital generierte Abbildungen verwendet. Im Gegensatz zu Thermobildern besteht zwischen akustischem Bild (Daten) und Original (Photo) meist nur sehr wenig optischer Zusammenhang.

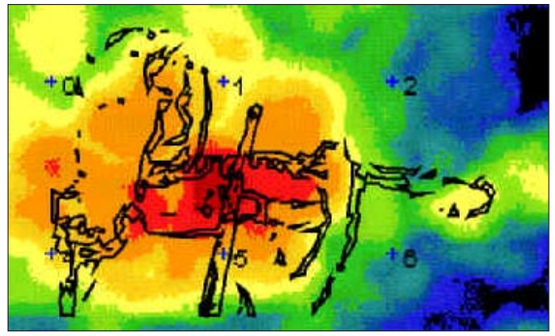
Um den örtlichen Bezug einer Schallursache zu erhalten, wird häufig auf Überlagerungsdarstellungen zurückgegriffen, in denen allerdings nur ein Bruchteil der ursprünglichen optischen Information sichtbar bleibt. Da das Farbspektrum, um eine maximale Differenzierung in der Darstellung zu bekommen, für jede Messung neu definiert wird, sind akustische Bilder nur schwer miteinander vergleichbar. Während die Farbe „Magenta“ in einer Abbildung nur 35 dBA repräsentiert, kann sie in einer anderen bereits 85 dBA bedeuten.



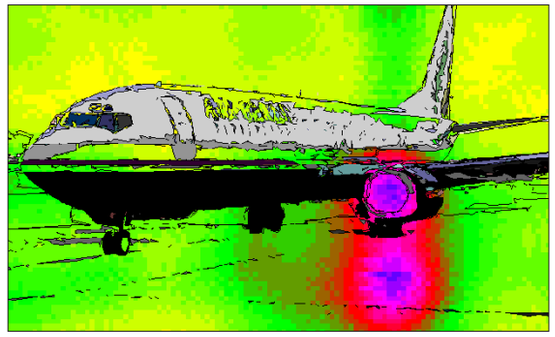
ICE bei ca. 200 km/h



Mercedes 230, Alpha Romeo, VW Passat bei ca. 50 km/h.



Geigenspielerin



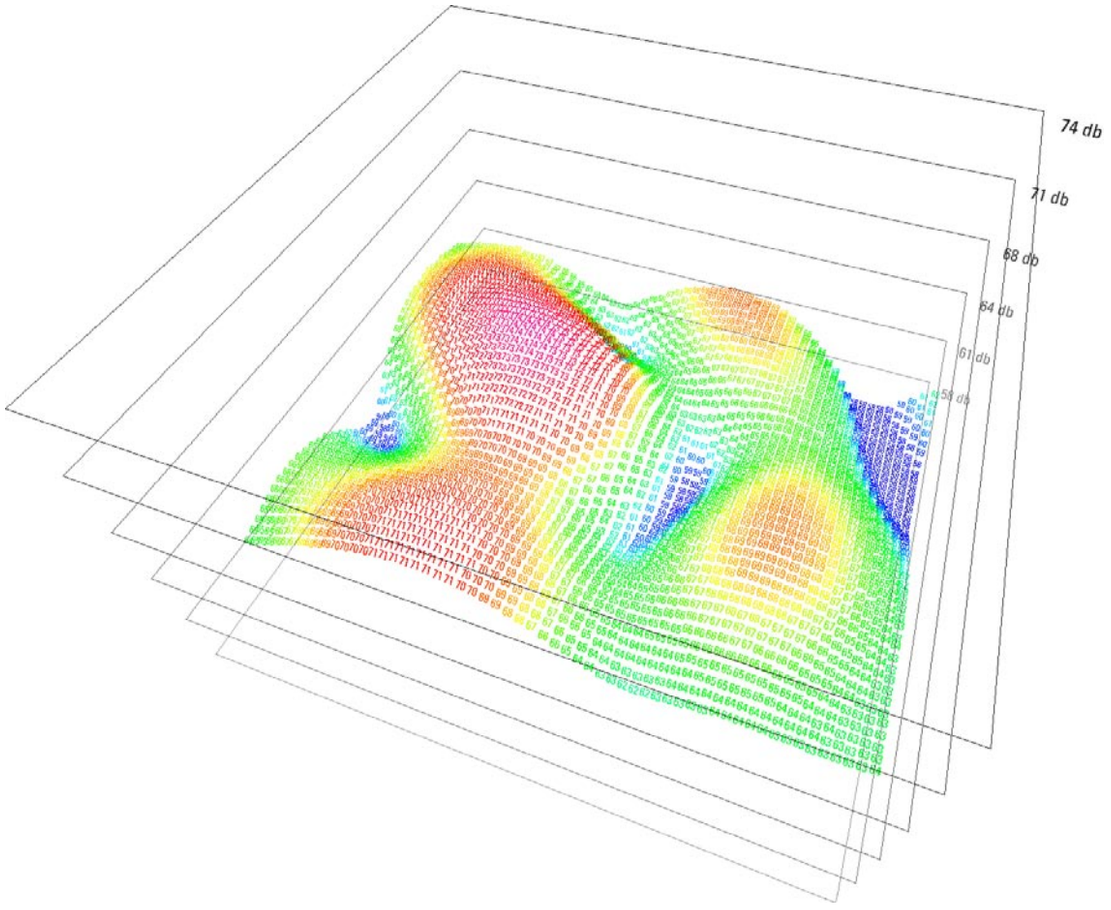
Boeing 737-400

```
// acoustic cartography ---------- // // 2004 daniel rothaug // mailto: daniel@acoustic-cartography.com // http://www.acoustic-cartoghraphy.com // // modified 2004-12-31 // p5 version: 74 int screenwidth = 800; int screenheight = 600-44; int imgheight = 350; // image heigth int imgwidth = 350; // image width int imgmargin = 43; // top and bottom image imgmargin int res = 10; // resolution int mode = 1; // default visualisation int fmode = 1; // default frequency mode int nmznum = -50; // default zoom int colormode = 2; // default colormode int min = 58; // min db value int max = 78; // max db value int offset; int count = 4; int dx; float d, h, nmx, nmy, nmz; float rxnum = 1.08; float rznum = -0.72; float rx, rz; float dnum = 1; PFont univers57, univers; boolean autorotate = false; boolean showimage = true; boolean showstage = true; boolean showdisplay = false; boolean showfilter = false; boolean showmarker = false; boolean showoutline = false; boolean showfrequencies = true; stage mystage = new stage(); display mydisplay = new display(); vimage vimage = new vimage(); int add = 1; int img_num = 11; frequencies[] img = new frequencies[img_num]; // setup ---------- void setup() { background(0); size(screenwidth, screenheight); // center nmx = width/2; nmy = height/2 + 60; univers57 = loadFont(„Univers_57_Condensed_14.vlw“); univers = loadFont(„Univers_57_Condensed.vlw“); // load texture image vimage.load(„texture.jpg“); // init image sequence for(int i=0; i<img.length; i+=“add)“ { img[i]=new frequencies(i); } // draw ---------- void draw() clear(); push(); transform(); if (autorotate) rznum +=“(width+1-dx)/1.5;“ (showimage || showmarker) vimage.update(); (showimage) vimage.render(); (showfrequencies) for(int i=“1;“ i<img.length; colorMode(HSB, 360); switch (fmode) case 1: img[i].vertexmap(color(0, 0, i*360/img.length, 80), false); break; 2: img[i].vertexmap(color(convertHSB(i, img.length), 360, 3: img[i].pixelmap(color(0, 200), 0); 4: img[i].outline(color(0, 180)); (showmarker) marker(100, 100); marker(200, marker(240, 240); marker(30, 50); 300); (showstage) mystage.scala(); (showfilter) mystage.layer(); pop(); display (showdisplay mouseX > width-20) show (abs(width-220-dx) 0.1) dx mydisplay.render(dx, else hide (abs(width+1-dx) 0);} ---------- functions 3d marker ---------- marker(int _x, int _y) (_y>imgmargin && _y < imgheight - imgmargin) { colorMode(HSB, 360); fill(convertHSB(count, img.length), 360, 360, 80); stroke(convertHSB(count, img.length), 360, 360); line(_x, _y, 0, _x, _y, vimage.p[_x][_y].t - 4); stroke(convertHSB(count, img.length), 360, 360, 80); line(_x, _y, vimage.p[_x][_y].t-4, 350, _y, vimage.p[_x][_y].t - 4); line(_x, _y, vimage.p[_x][_y].t-4, 0, _y, vimage.p[_x][_y].t - 4); line(_x, _y, vimage.p[_x][_y].t-4, _x, 0, vimage.p[_x][_y].t - 4); line(_x, _y, vimage.p[_x][_y].t-4, _x, 350, vimage.p[_x][_y].t - 4); push(); translate(_x, _y, vimage.p[_x][_y].t); box(8); textFont(univers, 10); fill(convertHSB(count, img.length), 360, 360); text(int(vimage.p[_x][_y].t*(max-min)/255) + min + „ db“, 10, 4); pop(); push(); translate(0, 0, vimage.p[_x][_y].t); noFill(); rect(0, 0, imgwidth, imgheight); pop(); } } // 3D transformation ---------- void transform() { if(mousePressed) { cursor(MOVE); if (abs(mouseX - nmx) > 0.01) { nmx += (mouseX-nmx)/6.0; } if (abs(mouseY - nmy) > 0.01) { nmy += (mouseY-nmy)/6.0; } } else { cursor(ARROW); } // zoom if (abs(nmz-nmznum) > 0.01) { nmz -= (nmz-nmznum)/2.0; } // center translate(nmx, nmy, nmz); // x-axis rotation if (abs(rx-rxnum) > 0.01) { rx -= (rx-rxnum)/2.0; }; rotateX(rx); // z-axis rotation if (abs(rz-rznum) > 0.01) { rz -= (rz-rznum)/2.0; }; rotateZ(rz); // center again translate(-imgwidth/2, -imgwidth/2); // amplitude if (abs(d-dnum) > 0.01) { d -= (d-dnum)/2.0; }; } // convert color values int convertHSB (int i, int nn) { int min = 240; // lowest value int max = 305; // highest value int range = 360; // spectrum range int c; c = i * ( range - (range-max) ) / nn + (range-min); if (c > range) { c -= range; } return range-c; // mirror and return value } // keyboard ---------- void keyPressed() { switch (keyCode) { // rotation case UP: rxnum += .18; break; case DOWN: rxnum -= .18; break; case RIGHT: rznum -= .18; break; case LEFT: rznum += .18; break; // filter case ALT: showfilter = !showfilter; break; case 34: showfilter = true; if (offset > 0) { offset -= 5; } break; case 33: showfilter = true; if (offset < 255 / 6 * 5) { offset += 5; } break; // frequencies (F1) case 112: fmode = 1; break; // frequencies (F2) case 113: fmode = 2; break; // frequencies (F3) case 114: fmode = 3; break; // frequencies (F4) case 115: fmode = 4; break; } switch (key) { // visualisations case ‚1‘: mode = 1; break; case ‚2‘: mode = 2; break; case ‚3‘: mode = 3; break; case ‚4‘: mode = 4; break; case ‚5‘: mode = 5; break; case ‚6‘: mode = 6; break; case ‚7‘: mode = 7; break; case ‚8‘: mode = 8; break; case ‚9‘: mode = 9; break; // adjust resolution case ‚+‘: if (res > 1) { res -= 1; } println(„resolution:\t“ + res); break; case ‚-‘: if (res < 30) { res += 1; }; println(„resolution:\t“ + res); break; case ‚c‘: if (colormode < 3) { colormode += 1; } else { colormode = 1; }; println(„colormode:\t“ + colormode); break; // save Frame case ‚s‘: saveFrame(); break; // zoom in case ‚:‘: nmznum += 40; break; // zoom out case ‚;‘: nmznum -= 40; break; // amplitude case ‚x‘: dnum += .2; break; case ‚y‘: dnum -= .2; break; // autorotate case ‚r‘: autorotate = !autorotate; break; // show / hide vimage case ‚i‘: showimage = !showimage; break; // show / hide frequencies case ‚f‘: showfrequencies = !showfrequencies; break; // show / hide db markers case ‚m‘: showmarker = !showmarker; break; // change frequency (space) case , ‚.‘: if (count < img_num-1) { count += 1; } else { count = 0; } println(„image:“ + count); break; // show / hide stage case ‚b‘: showstage = !showstage; break; // show / hide display case ‚d‘: showdisplay = !showdisplay; break; } } // ---------- // methods // ---------- // display ---------- class display { int ls = 28; // linespace int x, y; int nn; float ty; // frequencies in third-octave-bands String[] frequencies = {„500“, „630“, „800“, „1000“, „1250“, „1600“, „2000“, „2500“, „3150“, „4000“, „5000“}; String[] frequencies2 = {„447.0 - 562.0“, „562.0 - 708.0“, „708.0 - 891.0“, „891.0 - 1122.0“, „1122.0 - 1413.0“, „1413.0 - 1778.0“, „1778.0 - 2239.0“, „2239.0 - 2818.0“, „2818.0 - 3548.0“, „3548.0 - 4467.0“, „4467.0 - 5623.0“}; // render ---------- void render(float _x, float _y) { colorMode (HSB, 360, 100, 100, 100); x = int(_x); y = int(_y); overlay(x, 0, x, screenheight); title(x + 20, y + 40); //thumbnails(x + 20, y + 80); focus(x + 20 + 45, y + 500 + nn - 280); spectrum(x + 20 + 45, y + 500 - 280); } // background ---------- void overlay(int _x, int _y, int _w, int _h) { noStroke(); fill(0, 0, 0, 100); rect(_x, _y, screenwidth-_w, screenheight); stroke(0, 0, 360, 30); line(_x, _y, _x, screenheight); } // title ---------- void title(int _x, int _y) { textFont(univers57); textSpace(SCREEN_SPACE); textMode(ALIGN_LEFT); fill(0, 0, 100); fill(305, 100, 100); text(„acoustic cartography“, _x + 5, _y); fill(0, 0, 100); text(„frequency-analysis“, _x + 5, _y + 30); text(„of an automobile gear“, _x + 5, _y + ls * .75 + 30); } // thumbnails ---------- void thumbnails(int _x, int _y) { int w = img[count].img.width / 2; // thumbnail height int h = img[count].img.height / 2; // thumbnail width int offset = imgmargin-ls; // vertical space between thumbnails // frequency thumbnail image(img[count].img, _x, _y + h - offset, w, h); // mask fill(0,0,0); noStroke(); rect(_x, _y + h - offset, w, imgmargin/2); rect(_x, _y + h*2 - imgmargin/2 - offset, w, imgmargin/2); // frame noFill(); stroke(0,0,30); rect(_x, _y + h + imgmargin/2 - offset, w, h - imgmargin); textFont(univers57); textSpace(SCREEN_SPACE); textMode(ALIGN_LEFT); fill(0, 0, 20); text(„image „ + count + „ / „ + img_num, _x + 5, _y + h*2 - offset); // photographic thumbnail image(vimage.c, _x, _y, w, h); noFill(); stroke(0,0,30); rect(_x, _y + imgmargin/2, w, h - imgmargin); } // spectrum ---------- void spectrum(int _x, int _y) { textFont(univers57); textSpace(SCREEN_SPACE); textMode(ALIGN_RIGHT); for (int i=0; i<frequencies.length; i++) { int n=ls * (frequencies.length-i); fill(convertHSB(i, frequencies.length), 100, 100); text(frequencies[i] + „ Hz“, _x, _y n); if (i=“=“ !=“count)“ fill(0, 0, 30); } text(frequencies2[i] _x 125, count) nn=“n;“ // focus ---------- void focus(int _y) (abs(ty ->0.01) { ty -= (ty - _y) / 1.5; }; int x = _x - 45; float y = ty - 14; noStroke(); fill(0, 0, 10); rect(x, y, 176, 18); } } // frequencies ---------- class frequencies { PImage img; int[][] imgColors = new int[imgwidth][imgheight]; int[][] imgPixels = new int[imgwidth][imgheight]; float t, tt; int id; int hshift = 175; // HSB offset frequencies(int _id) { id = _id; loadTexture(); } // load texture image ---------- void loadTexture() { img = loadImage(„img_“ + id + „.jpg“); println(„img_“ + id + „.jpg \t loaded“); // analyse(); } // pixelmap ---------- void pixelmap(color _c, int _offset) { colorMode(RGB, 255); for(int i=0; i<imgwidth-res; i+=res) { for(int j=imgmargin; j<imgheight-imgmargin; j+=“res)“ { if (brightness(imgPixels[j][i])>160) { stroke(_c); point (i, j, imgColors[j][i] * d + _offset); } } } } // image outline ---------- void outline(color _c) { beginShape(LINE_STRIP); stroke(_c); // left for(int j=imgmargin; j<=imgheight-imgmargin-1; j+=res) { int i = 0; vertex(vimage.p[i][j].x, vimage.p[i][j].y, imgColors[j][i] * d); } // bottom for(int i=0; i<imgwidth; i+=“res)“ { int j=“imgheight-imgmargin-1;“ -imgmargin 1; vertex(vimage.p[i][j].x, vimage.p[i][j].y, imgColors[j][i] * d); } // right for(int>=imgmargin; j-=res) { int i = imgwidth - 1; vertex(vimage.p[i][j].x, vimage.p[i][j].y, imgColors[j][i] * d); } // top for(int i=0; i<imgwidth; i+=res) { int j = imgmargin; int ii = imgwidth - i - 1; vertex(vimage.p[ii][j].x, vimage.p[ii][j].y, imgColors[ii][j] * d); } vertex(vimage.p[0][imgmargin].x, vimage.p[0][imgmargin].y, imgColors[0][imgmargin] * d); endShape(); } // vertexmap ---------- void vertexmap(color _c, int _offset, boolean _texture) { colorMode(RGB, 255); for(int i=0; i<imgwidth-res; i+=res) { if (_texture) { beginShape(TRIANGLE_STRIP); fill(255); } else { beginShape(LINE_STRIP); fill(0); } for(int j=imgmargin; j<imgheight-imgmargin; j+=res) { t = imgColors[j][i] * d; tt = imgColors[j][i+res] * d; stroke(_c); texture(img); vertex(i,j,t + _offset,i,j); vertex(i+res, j, tt + _offset, i+res, j); } endShape(); } } // analyse image ---------- void analyse() { for(int i=0; i<imgwidth; i++) { for(int j=0; j<imgheight; j++) { imgPixels[i][j]=“img.pixels[i*imgheight+j];“ h=“hue(img.pixels[i*imgheight+j]);“ if (h>hshift) { h = h - 255; } imgColors[i][j] = int(hshift - h); } } } // stage ---------- class stage { int n = 6; // scala float off; // filter position // box ---------- void scala() { colorMode(RGB, 255); textFont(univers, 18); textSpace(OBJECT_SPACE); textMode(ALIGN_LEFT); rectMode(CORNER); for(int i = 0; i < n; i++) { push(); translate(0, 0 ,i * 255/n * d); fill(255, i * 100/n + 50); text(int(i*(max-min)/n) + min + „ db“, imgwidth+10 , 18/2); noFill(); stroke(255, i * 100/n + 50); rect(0, 0, imgwidth, imgheight); pop(); } } // db layer ---------- void layer() { if (abs(off-offset) > 0.01) { off -= (off-offset)/1.5; }; push(); translate(0, 0, off * d); fill(255); textSpace(OBJECT_SPACE); textFont(univers, 22); text(int(off*(max-min)/255) + min + „ db“, imgwidth + 10 , imgheight); stroke(255); fill(0, 200); rect(0, 0, imgwidth, imgheight); pop(); } } // imagemap ---------- class vimage { PImage c; int[][] cPixels = new int[imgwidth][imgheight]; particle[][] p = new particle[imgwidth][imgheight]; float tt, tnum; String name; // init particles ---------- vimage() { for(int i=0; i<imgwidth; i++) { for(int j=imgmargin; j<imgheight-imgmargin; j++) { p[i][j] = new particle(i, j); } } } // load image ---------- void load(String _name) { name = _name; c = loadImage(name); println (name + „\t loaded“); analyse(); } // analyse texture ---------- void analyse() { for(int i=0; i<imgwidth; i++) { for(int j=0; j<imgheight; j++) { cPixels[j][i] = c.pixels[i*imgheight+j]; } } } // render ---------- void render() { switch (mode) { case 1: vertexmap(true); break; case 2: pixelmap(1); break; case 3: pixelmap(2); break; case 4: pixelmap(3); break; case 5: pixelmap(4); break; case 6: vertexmap(true); pixelmap(2); break; case 7: pixelmap(3); pixelmap(2); break; case 8: pixelmap(5); break; case 9: break; } if (showoutline) { outline(); } } // pixelmap ---------- void pixelmap(int _mode) { colorMode(RGB, 255); for(int i=0; i<imgwidth; i+=res) { for(int j=imgmargin; j<imgheight-imgmargin; j+=res) { switch (_mode) { // visualisation: points ---------- case 1: switch (colormode) { case 1: stroke (255, 255, 255, 200); break; case 2: stroke(red(cPixels[i][j]), green(cPixels[i][j]), blue(cPixels[i][j])); break; case 3: stroke(red(img[count].imgPixels[j][i]), green(img[count].imgPixels[j][i]), blue(img[count].imgPixels[j][i])); break; } point(p[i][j].x, p[i][j].y, p[i][j].t); break; // visualisation: lines ---------- case 2: switch (colormode) { case 1: stroke (255, 255, 255, 50); break; case 2: stroke(red(cPixels[i][j]), green(cPixels[i][j]), blue(cPixels[i][j]), 200); break; case 3: stroke(red(img[count].imgPixels[j][i]), green(img[count].imgPixels[j][i]), blue(img[count].imgPixels[j][i]), 200); break; } if (count < img_num - 1) { tnum = img[count + 1].imgColors[j][i] * d; } else { tnum = img[0].imgColors[j][i] * d; } if (abs(tt - tnum) > 0.01) { tt -= (tt - tnum)/2.0; } line(p[i][j].x, p[i][j].y, p[i][j].t, p[i][j].x, p[i][j].y, tt); break; // visualisation: cubes ---------- case 3: rectMode(CORNER); switch (colormode) { case 1: fill(255, 255, 255, 30); stroke (255, 255, 255, 50); break; case 2: fill(red(cPixels[i][j]), green(cPixels[i][j]), blue(cPixels[i][j])); stroke(0, 50); break; case 3: fill(red(img[count].imgPixels[j][i]), green(img[count].imgPixels[j][i]), blue(img[count].imgPixels[j][i])); stroke(0, 50); break; } push(); translate(p[i][j].x, p[i][j].y, p[i][j].t); box(res-2); pop(); break; // visualisation: graph ---------- case 4: rectMode(CORNER); switch (colormode) { case 1: fill(255, 255, 255, 30); stroke (255, 255, 255, 50); break; case 2: fill(red(cPixels[i][j]), green(cPixels[i][j]), blue(cPixels[i][j])); stroke(0, 50); break; case 3: fill(red(img[count].imgPixels[j][i]), green(img[count].imgPixels[j][i]), blue(img[count].imgPixels[j][i])); stroke(0, 50); break; } push(); translate(p[i][j].x, p[i][j].y, p[i][j].t/2); box(res-2, res-2, p[i][j].t); pop(); break; // visualisation: typography ---------- case 5: noStroke(); switch (colormode) { case 1: fill(255, 255, 100); break; case 2: fill(red(cPixels[i][j]), green(cPixels[i][j]), blue(cPixels[i][j]), 200); break; case 3: fill(red(img[count].imgPixels[j][i]), green(img[count].imgPixels[j][i]), blue(img[count].imgPixels[j][i]), 200); break; } textFont(univers, 8); textSpace(OBJECT_SPACE); textMode(ALIGN_LEFT); text(int(p[i][j].t * (max - min)/255) + min, p[i][j].x, p[i][j].y, p[i][j].t); break; } } } } // vertexmap ---------- void vertexmap(boolean _texture) { colorMode(RGB, 255); switch (colormode) { case 1: fill(255, 0); stroke(255, 50); break; case 2: fill(255); stroke(0, 50); break; case 3: fill(255); stroke(0, 50); break; } for(int i=0; i<imgwidth-res; i+=res) { beginShape(TRIANGLE_STRIP); if (colormode == 3) { texture(img[count].img); } else { texture(c); } for(int j=imgmargin; j<imgheight-imgmargin-res-1; j+=res) { vertex(p[i][j].x, p[i][j].y, p[i][j].t, p[i][j].x, p[i][j].y); vertex(p[i+res][j].x, p[i+res][j].y, p[i+res][j].t, p[i+res][j].x, p[i+res][j].y); } endShape(); } } // outline ---------- void outline() { colorMode(HSB, 360); /* switch (colormode) { case 1: stroke(0, 0, 360, 120); break; case 2: stroke(0, 0, 360, 120); break; case 3: stroke(convertHSB(count, img.length), 360, 360); break; } */ beginShape(LINE_STRIP); // left for(int j=imgmargin; j<=imgheight-imgmargin-1; j+=res) { int i = 0; vertex(p[i][j].x, p[i][j].y, p[i][j].t); } // bottom for(int i=0; i<imgwidth; i+=“res)“ { int j=“imgheight-imgmargin-1;“ - imgmargin 1; vertex(p[i][j].x, p[i][j].y, p[i][j].t); } // right for(int>=imgmargin; j-=res) { int i = imgwidth - 1; vertex(p[i][j].x, p[i][j].y, p[i][j].t); } // top for(int i=0; i<imgwidth; i+=res) { int j = imgmargin; int ii = imgwidth - i - 1; vertex(p[ii][j].x, p[ii][j].y, p[ii][j].t); } vertex(p[0][imgmargin].x, p[0][imgmargin].y, p[0][imgmargin].t); endShape(); } // update points ---------- void update() { for(int i=0; i<imgwidth; i++) { for(int j=imgmargin; j<imgheight-imgmargin; j++) { p[i][j].update(count); } } } // reference points ---------- class particle { int x, y; float t, tn; particle(int _x, int _y) { x = _x; y = _y; } void update(int _count) { tn = img[_count].imgColors[y][x] * d; if (abs(t-tn) > 0.01) { t -= (t-tn)/2.0; }; } void render(int _count) { colorMode(RGB, 255); stroke(255); point(x, y, t); } } }
```

Technische Vorraussetzung für die Visualisierungen ist eine in Processing/JAVA entwickelte Software. Sie ermöglicht es, die Spektralbilder der „Akustischen Kamera“ zu analysieren und nach neuen, frei definierten Kriterien wieder zusammenzusetzen. Dadurch konnte eine einheitliche Datengrundlage geschaffen werden, die es erlaubt, unterschiedliche akustische Szenarien abzubilden und miteinander zu vergleichen.

Die Anwendung wurde in Processing (Alpha) 69 und 74 realisiert. Processing ist eine Open-Source Programmierumgebung, die zur Programmierung digitaler Prototypen und Entwürfe im Designbereich entwickelt wird.

Weitere Informationen:
www.acoustic-cartography.com/processing/
www.processing.org



Als Grundlage dieser Arbeit dienen Aufzeichnungen der „Akustischen Kamera“ in Form von Einzelbildern oder Bildsequenzen. Im ersten Schritt wird das Bildmaterial durch die Software zweidimensional analysiert. Aus jedem einzelnen Pixel wird der abgebildete Farbwert (HSB) extrahiert. Dadurch kann der ursprüngliche Messwert (mit gewissen Unschärfen) an diesem Bildpunkt rekonstruiert werden.

Die ausgelesenen Werte dienen als Verzerrungsmatrix, die das akustische Spektralbild in ein dreidimensionales Höhenrelief übersetzt. Anstelle des akustischen Bildes kann nun eine Textur des Originalbildes über das Relief gelegt werden. Zur besseren räumlichen Orientierung ist das Relief in eine dreidimensionale Bühnenfläche mit Skala eingeschlossen. Die Bühne kann frei rotiert, verschoben und vergrößert werden. Weitere Filterebenen können eingeblendet werden und erleichtern die Betrachtung.

Nach diesem System wurden verschiedene, experimentelle Visualisierungsmodelle entwickelt, die es ermöglichen akustische und optische Informationen in einer Abbildung gleichzeitig sichtbar zu machen.

F1 - F3
Frequenz-Visualisierungen

1 - 9
Visualisierungen

I
Bildrelief
an / aus

Plus
Auflösung
verstärken

Return
nächstes
Einzelbild

Page Up
Filter-Ebene
aufwärts

Page Down
Filter-Ebene
abwärts

D
Display
an / aus

F
Frequenzen
an / aus

Y
Amplitude
verstärken

X
Amplitude
verringern

C
Farbmodus
wechseln

B
Bühne
an / aus

M
Marker
an / aus

Komma
einzoomen

Punkt
auszoomen

Minus
Auflösung
verringern

Oben

Links **Unten** **Rechts**
Bühne rotieren

Alt
Filter-Ebene
an / aus

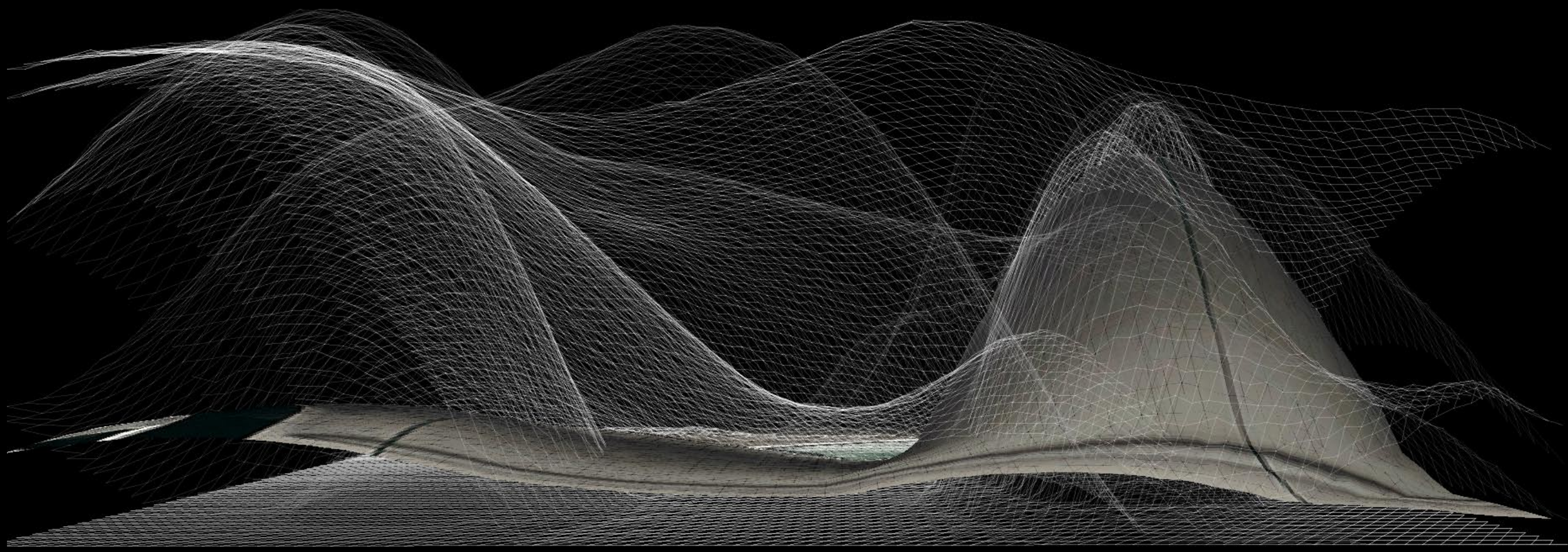
Space
nächste Präsentationsansicht

Mouse klicken und ziehen
Bühne bewegen

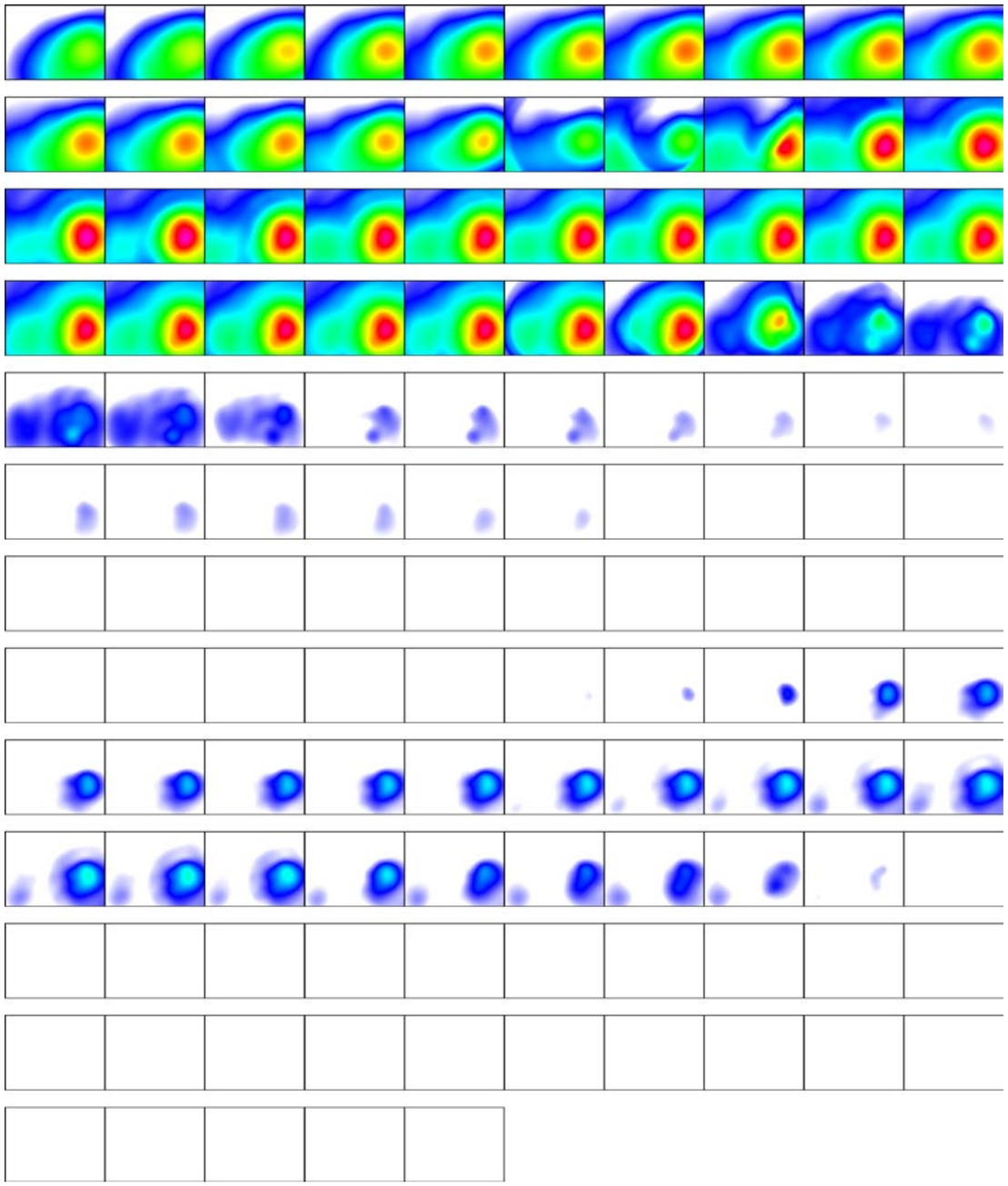
AKUSTISCHE KINETIK

Die zeitliche (kinetische) Veränderung des Schalls beim Türschlag eines Automobils in 62 ms. Durch Überlagerungen und Verdichtungen werden schnelle und langsame Veränderungen sichtbar.

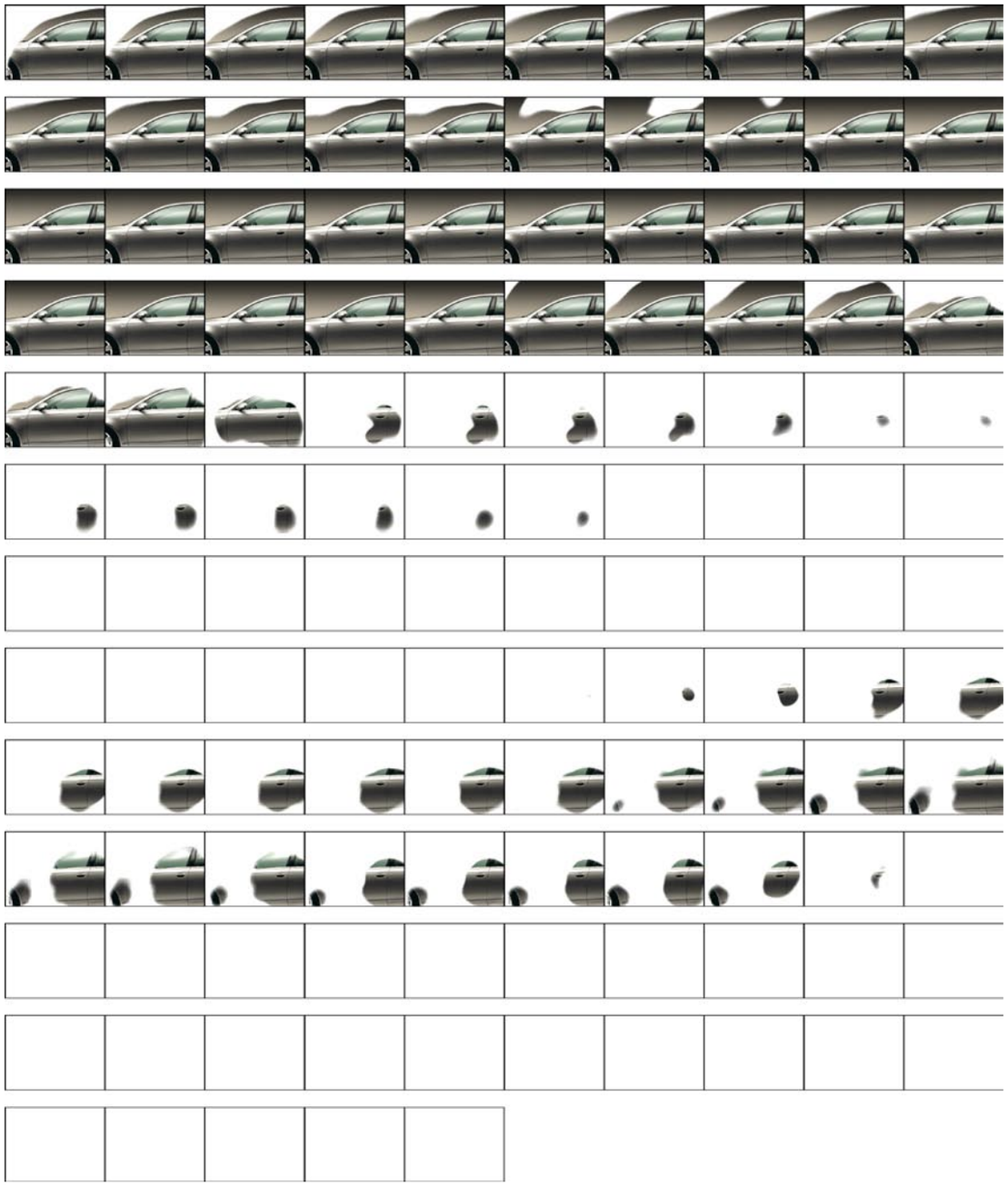
Für die Visualisierung wurden 124 Einzelbilder mit insgesamt 11.457.600 Bildpunkten (Pixel) analysiert, verarbeitet und abgebildet.



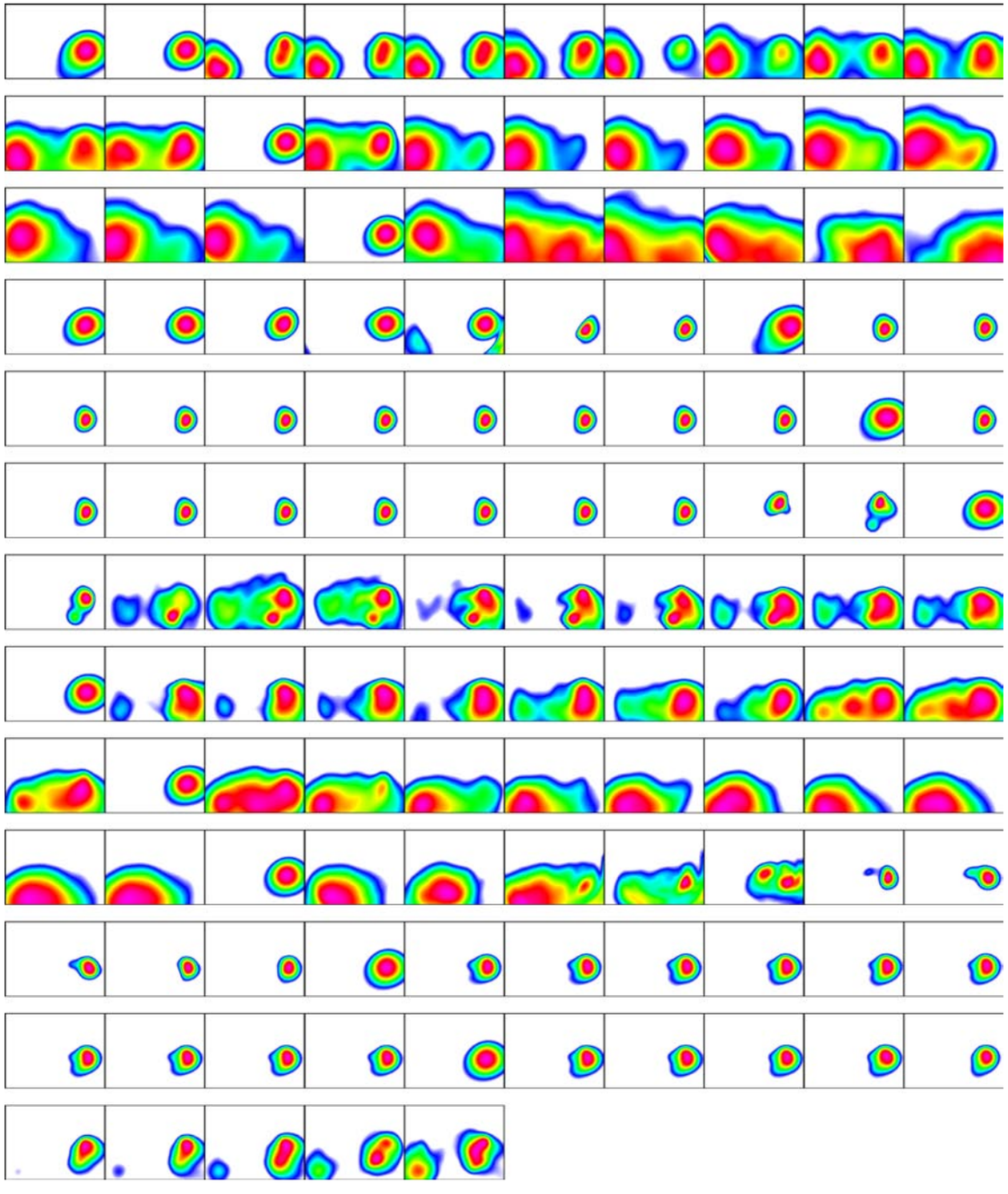
Ausgangsmaterial ist eine Bildsequenz von 62 ms aus 124 Einzelbildern. Ein Einzelbild entspricht 0,5 ms, eine Zeile 5 ms. Die Sequenz zeigt den Türschlag einer Automobiltür bei einem konstanten (unfokussierten) dB-Bereich von 75 - 85 dBA.



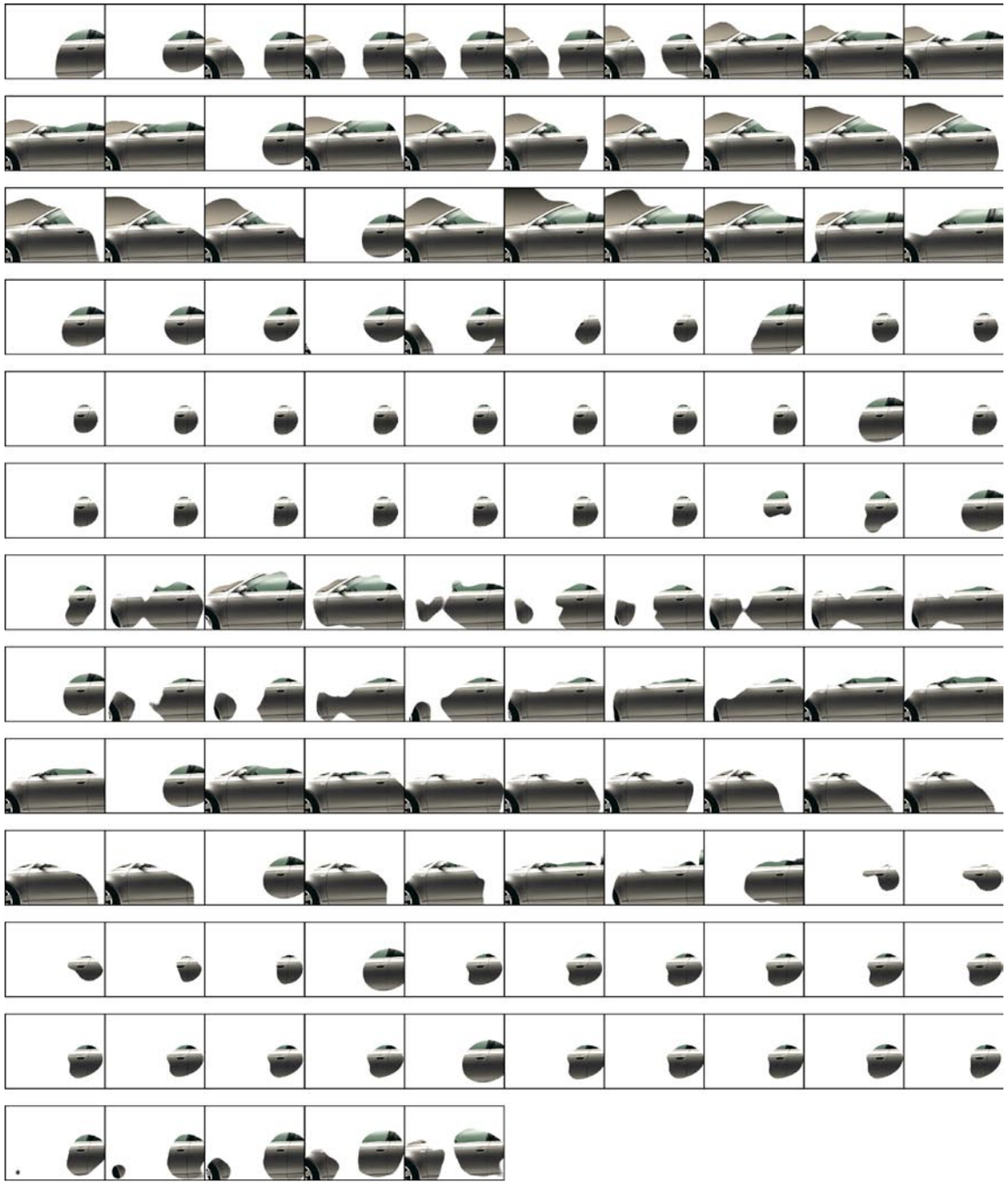
Photographischer Kontaktabzug der gleichen Bildsequenz. Nur Bereiche innerhalb des konstanten dB-Bereichs werden abgebildet.

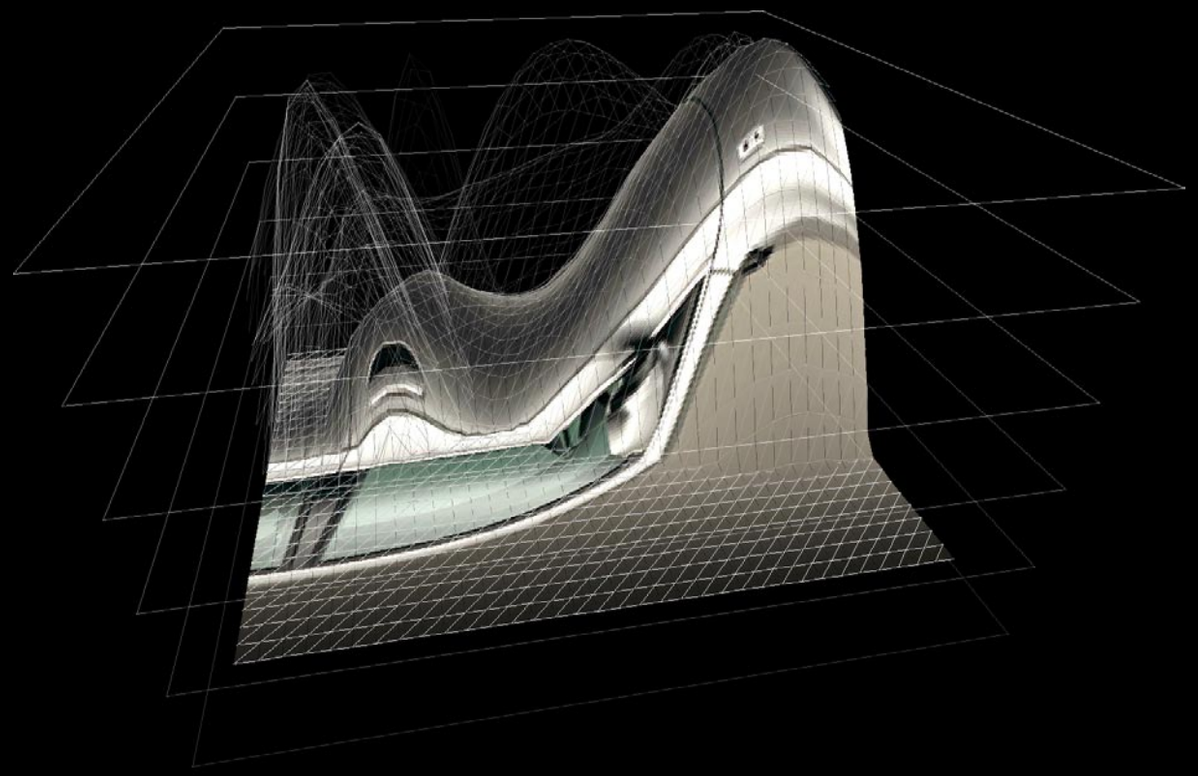


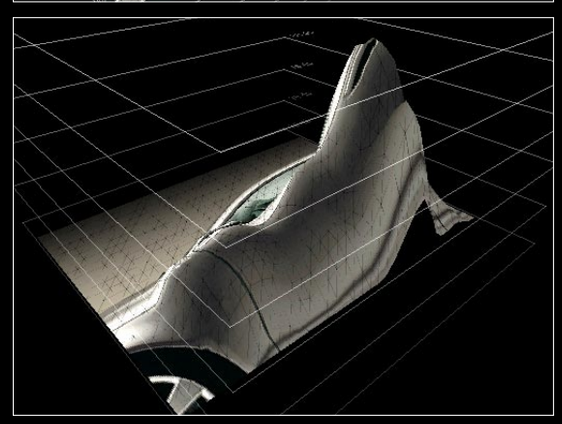
Die Sequenz mit variablem (fokussiertem) dB-Bereich für jedes Einzelbild. So können detaillierte Veränderungen festgestellt werden. Die Werte des HSB-Farbspektrums sind unter den Einzelbildern jedoch nicht mehr vergleichbar.

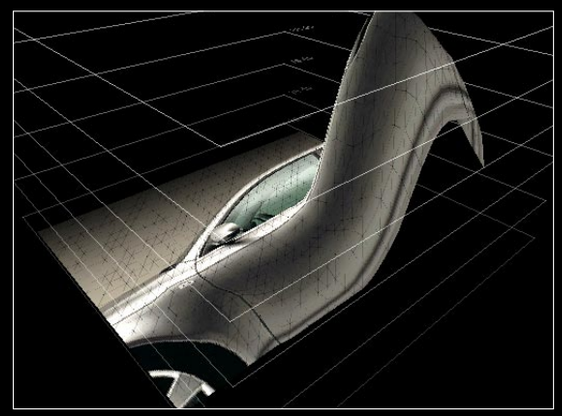


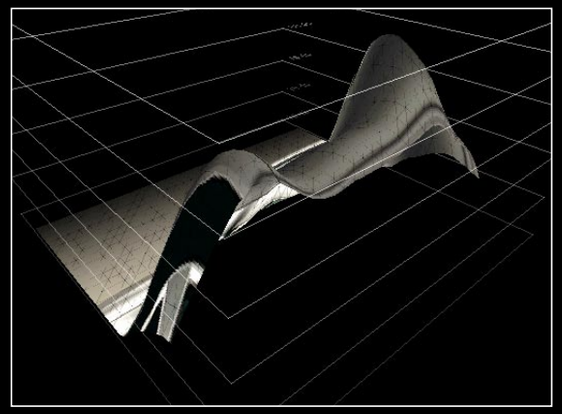
Photographischer Kontaktabzug der gleichen Bildsequenz. Nur Flächen innerhalb des variablen dB-Bereichs werden abgebildet.

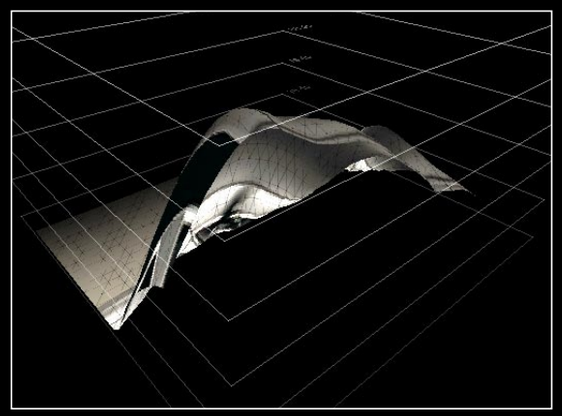


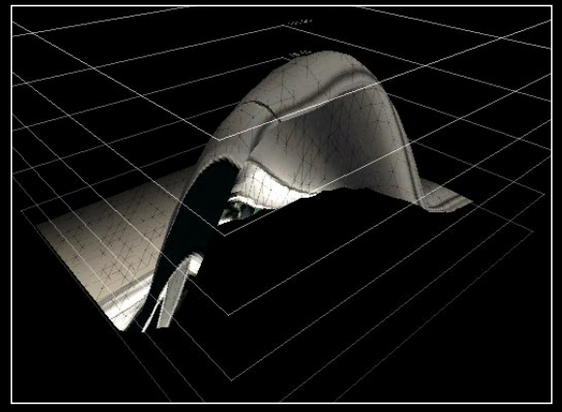


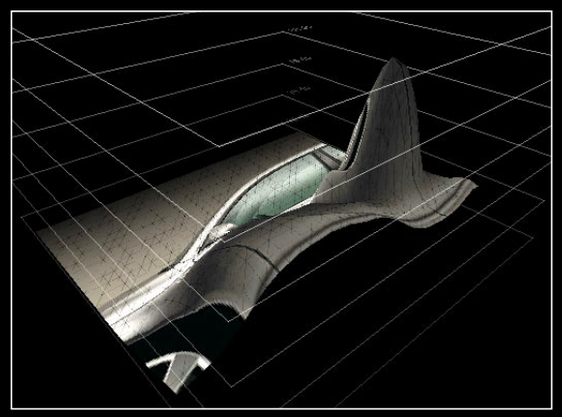


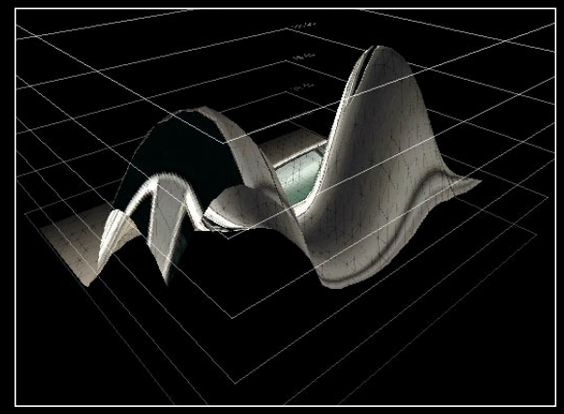


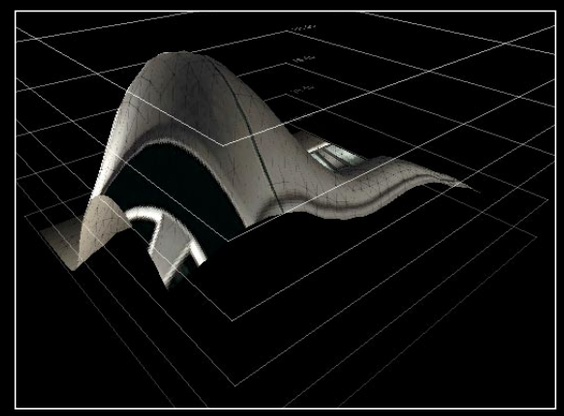


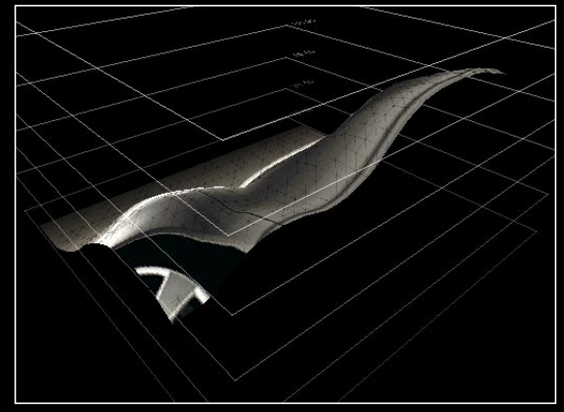


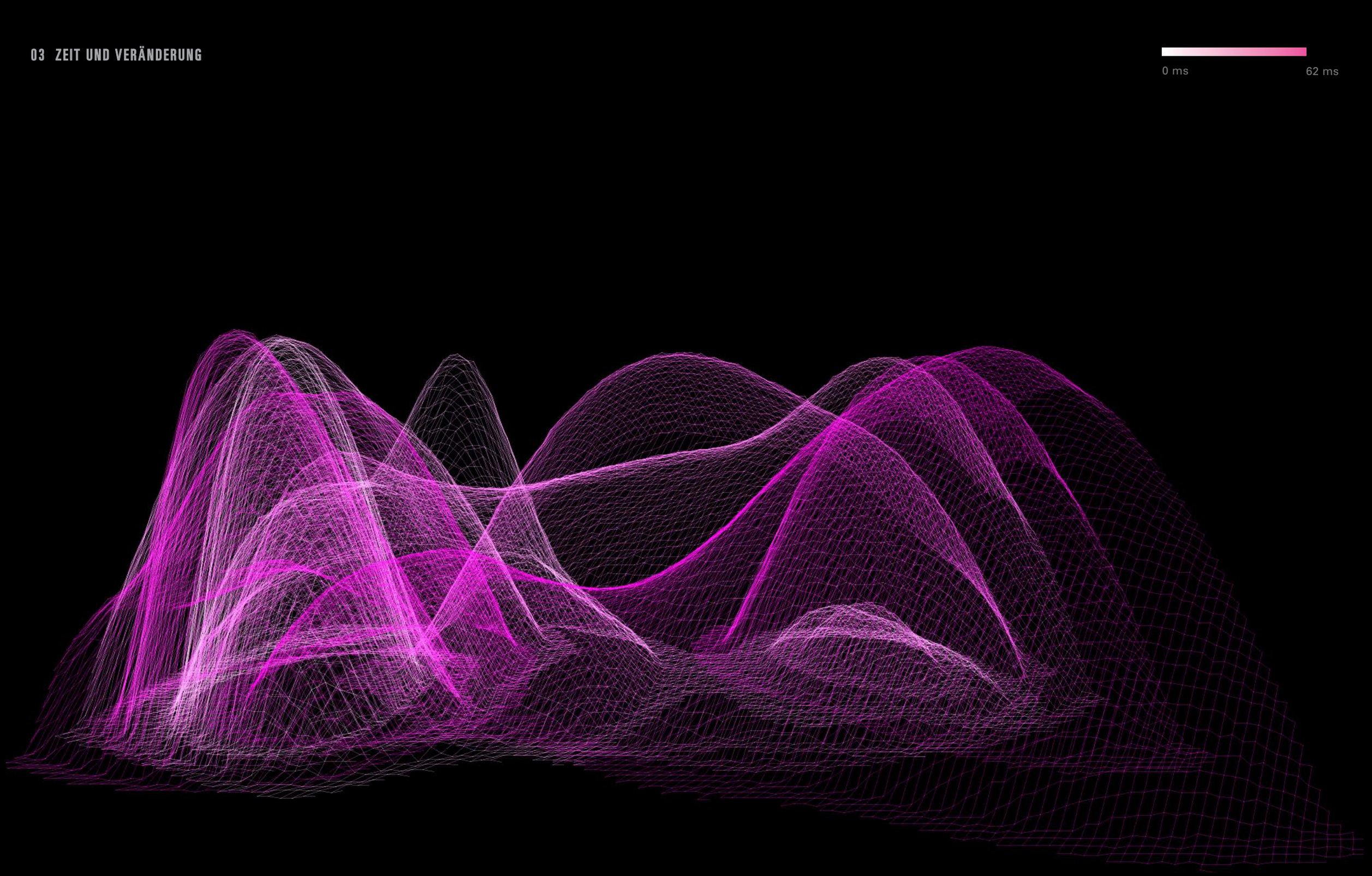






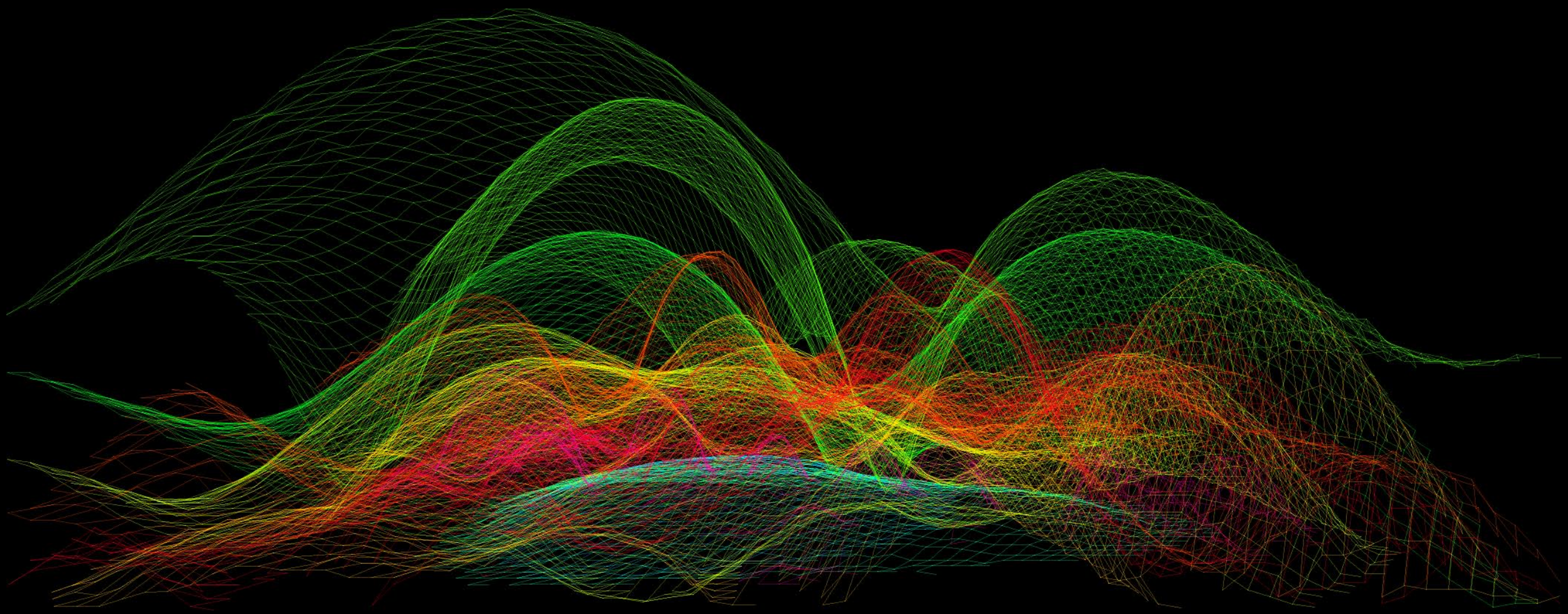





0 ms
62 ms

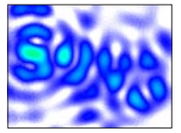
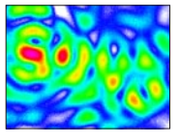
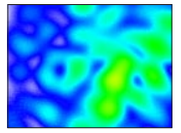
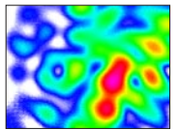
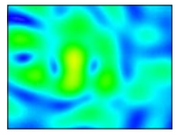
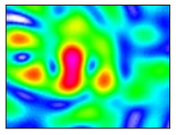
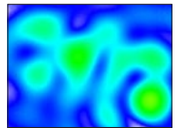
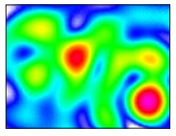
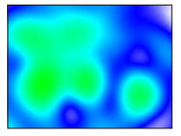
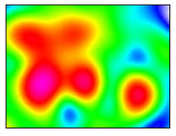
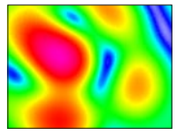
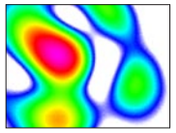
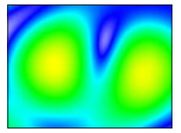
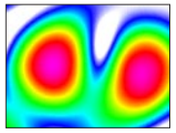
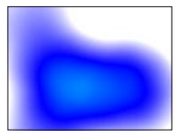
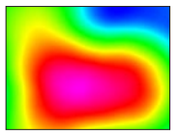
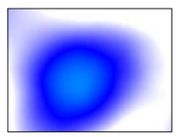
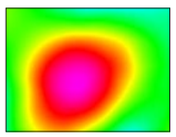
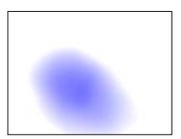
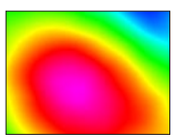
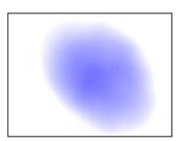
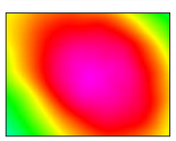
Bisher wurden ausschließlich Lautstärke und Zeit abgebildet. Eine weitere, entscheidende Rolle spielt das Lautstärkevorkommen innerhalb des Frequenzspektrums. Durch das typische Schwingverhalten bestimmter Materialien können Frequenzen (z.B. in der Fehleranalyse) Hinweis auf mögliche Ursachen geben.

Die Visualisierung zeigt die Frequenzverteilung eines Getriebes. Die Frequenzen sind in Terzbänder von 500 - 5000 Hz in einem Bereich von 58 - 74 dB gegliedert. Der Farbcode entspricht dem HSB-Farbspektrum.

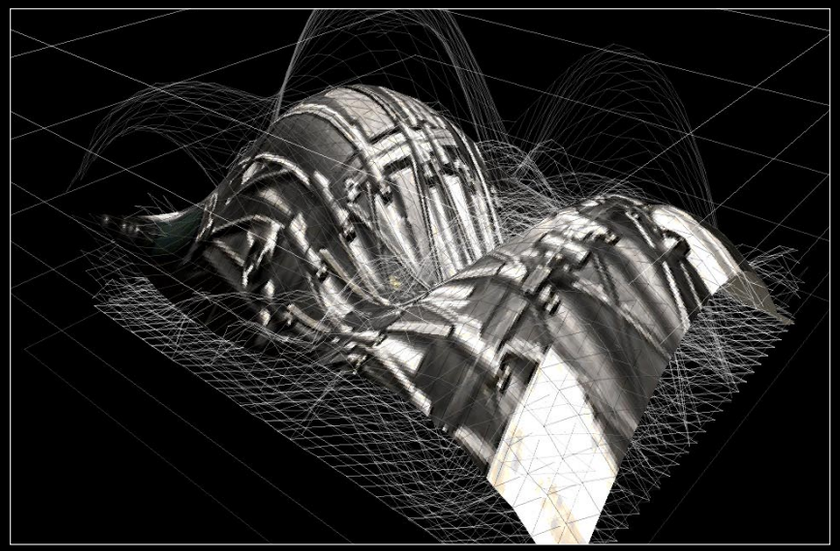


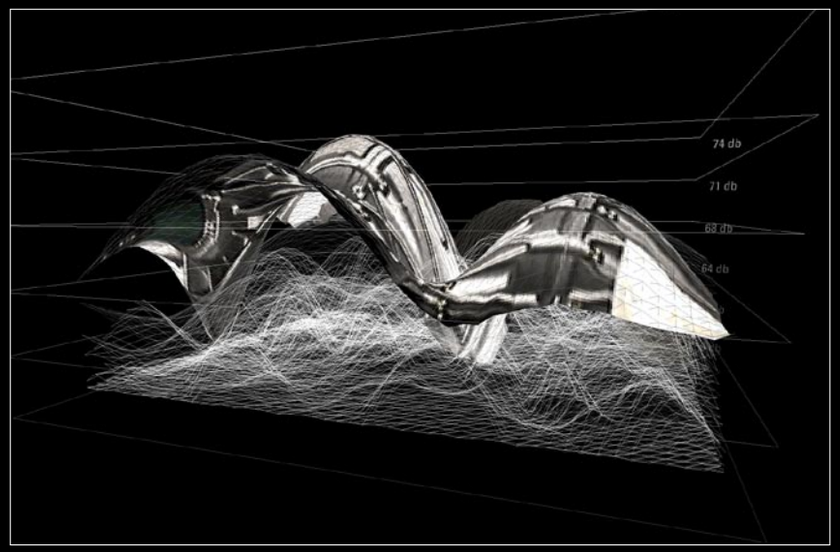
Ausgangsmaterial für diese Visualisierung ist eine Bildsequenz, bei der für jedes Frequenzband ein Spektralbild verwendet wird. Die Farbcodierung ist mit Frequenzen und dBA-Werten doppelt belegt.

Bisher werden diese Aufzeichnungen als Film interpretiert, d.h. die Gleichzeitigkeit der Frequenzen wird dabei übergangen. Durch die mehrdimensionale Darstellung ist nun eine synchrone Abbildung möglich. Der HSB-Farbcode kann ausschließlich für die Frequenzbänder verwendet werden.

Konstante Fokussierung auf 58 - 74 dBA	Variable Fokussierung des dBA-Bereichs	Frequenzverteilung in Terzbändern
		**5000 Hz** 4467.0 - 5623.0
		**4000 Hz** 3548.0 - 4467.0
		**3150 Hz** 2818.0 - 3548.0
		**2500 Hz** 2239.0 - 2818.0
		**2000 Hz** 1778.0 - 2239.0
		**1600 Hz** 1413.0 - 1778.0
		**1250 Hz** 1122.0 - 1413.0
		**1000 Hz** 891.0 - 1122.0
		**800 Hz** 708.0 - 891.0
		**630 Hz** 562.0 - 708.0
		**500 Hz** 447.0 - 562.0

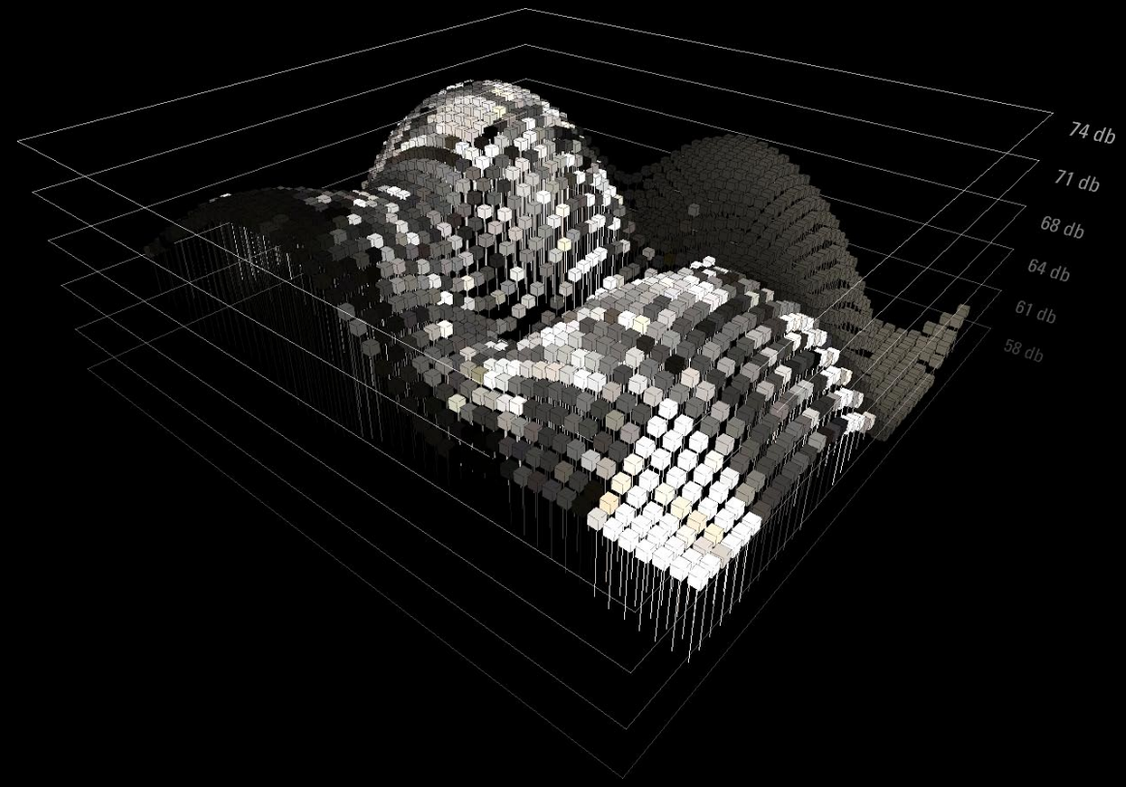
01 BILDTEXTUR MIT FREQUENZEN



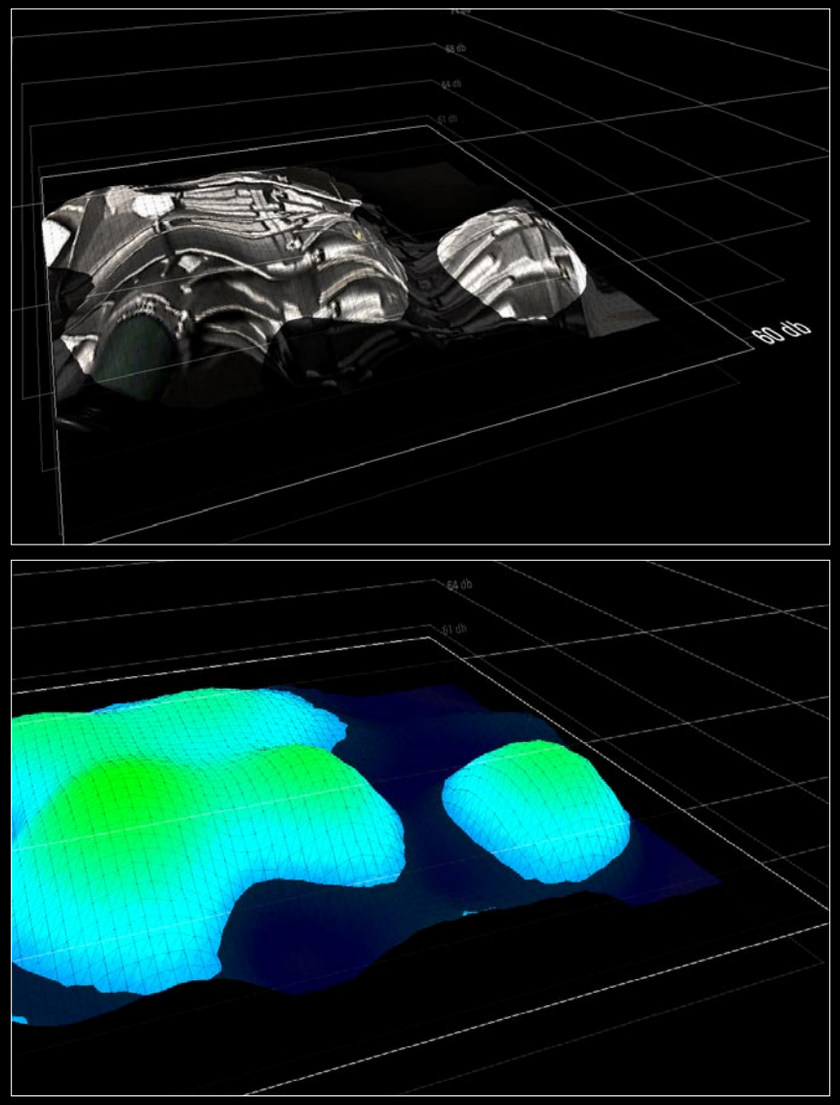


02 FREQUENZ-VERGLEICH

Mit Vektor-Linien kann der Unterschied zwischen zwei akustischen Situationen dargestellt werden. Die Würfel zeigen die aktuellen Werte.



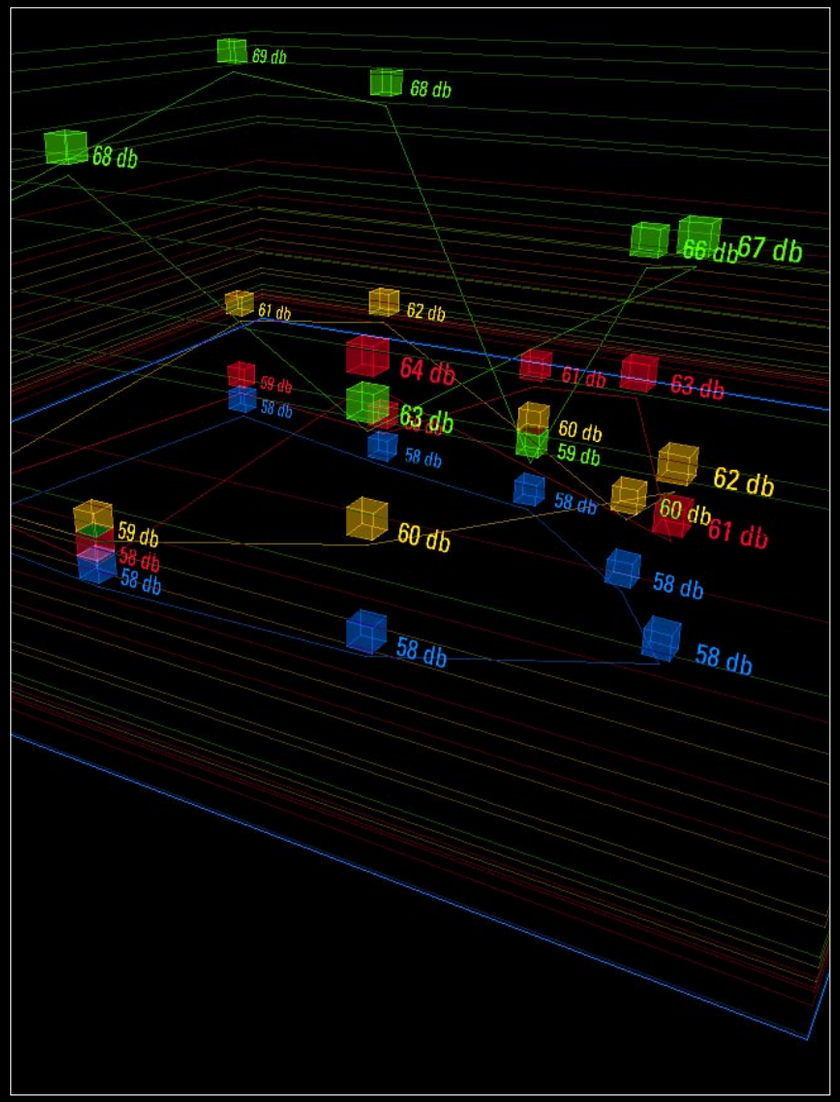
Bestimmte Werte-Bereiche können durch zusätzliche, frei verschiebbare Filterebenen besonders hervorgehoben werden. So sind z.B. Pegelüberschreitungen eines Maximalwertes besser sichtbar. Filter sind in beliebiger Kombination und Anzahl auch für weitere Aufgaben denkbar.

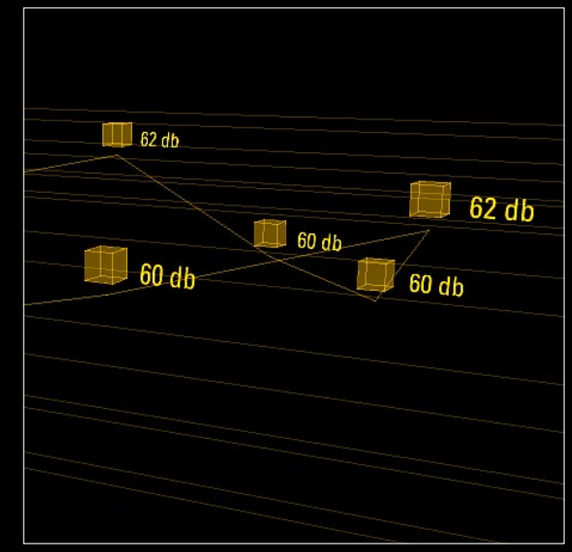


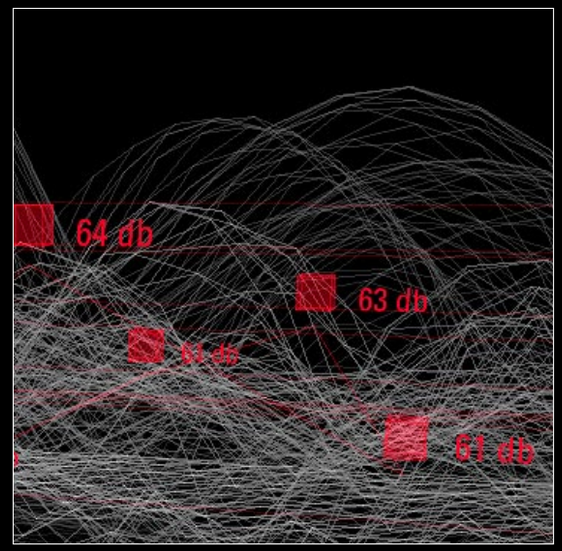
Die Messwerte sind, typographisch umgesetzt, direkt an ihren Koordinaten ablesbar.

05 MARKER

Um gezielt einzelne Punkte (z.B. eine besondere Verarbeitungsstelle) zu beobachten, können zusätzliche Hilfs-Markierungen (Marker) gesetzt werden, die Werte bestimmter Koordinaten besonders kennzeichnen. Die Marker können mit anderen Visualisierungen kombiniert werden. Sie zeigen durch den Farbcode ihren entsprechenden Frequenzbereich.

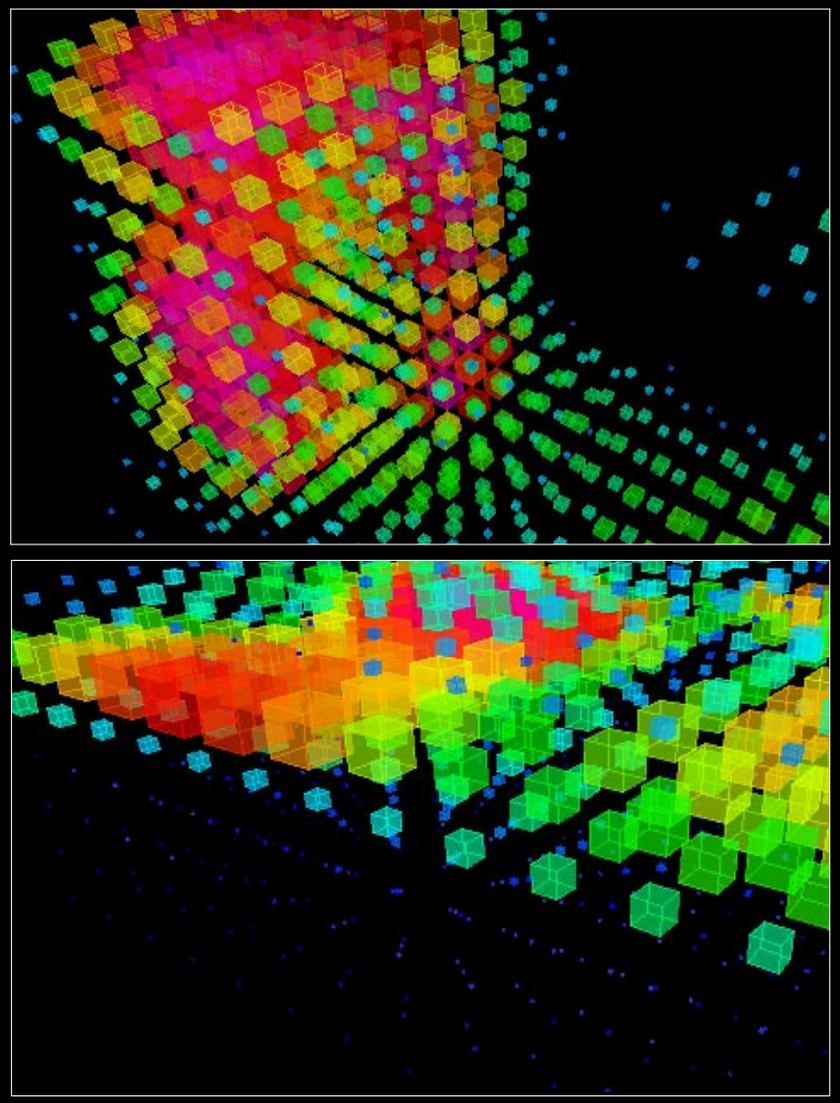


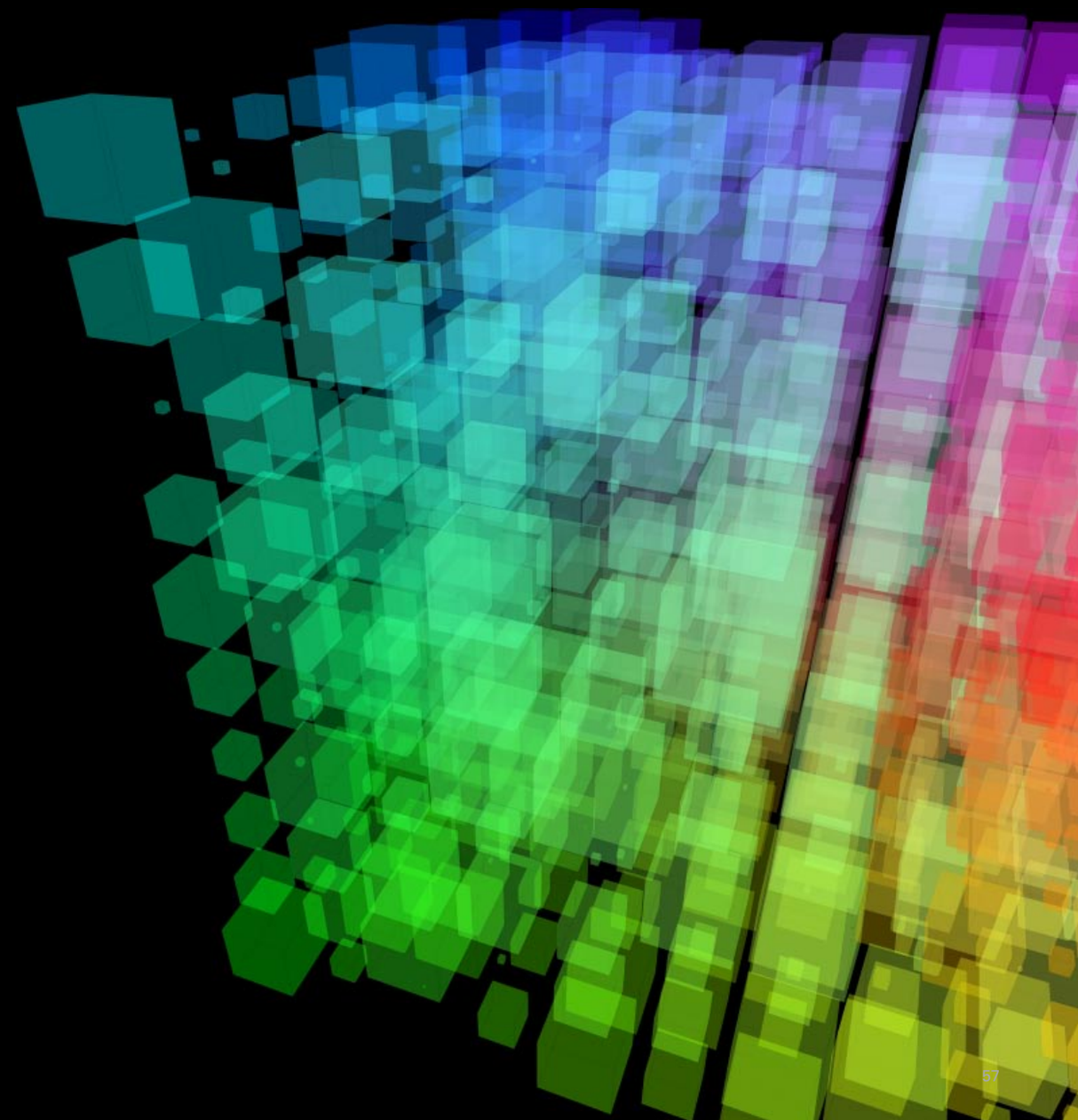




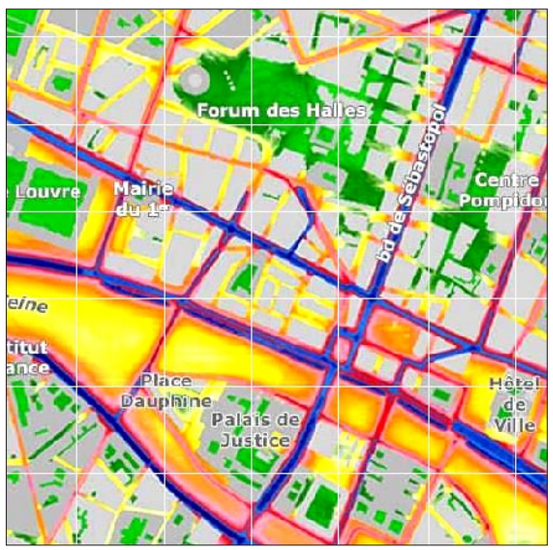
DREIDIMENSIONALE KARTOGRAPHIE

Eine weitere Herausforderung sind holographische oder dreidimensionale Schallabbildungen, da die räumliche Dimension zusätzlich berücksichtigt werden muss. Mit simulierten Daten konnten erste Experimente realisiert werden.





www.paris.fr/fr/environnement/bruit/



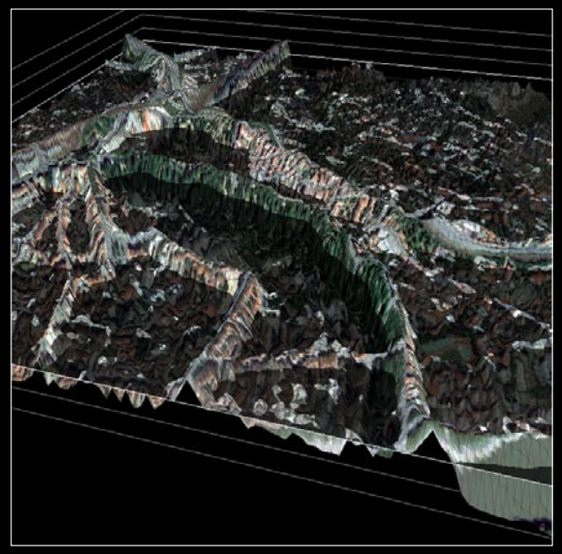
Auch bei Lärmkarten, die aus einer Kombination von Messungen und Berechnungen erstellt werden, zeigt sich in der Datenabbildung ein ähnliches Problem wie z.B. bei der „Akustischen Kamera".

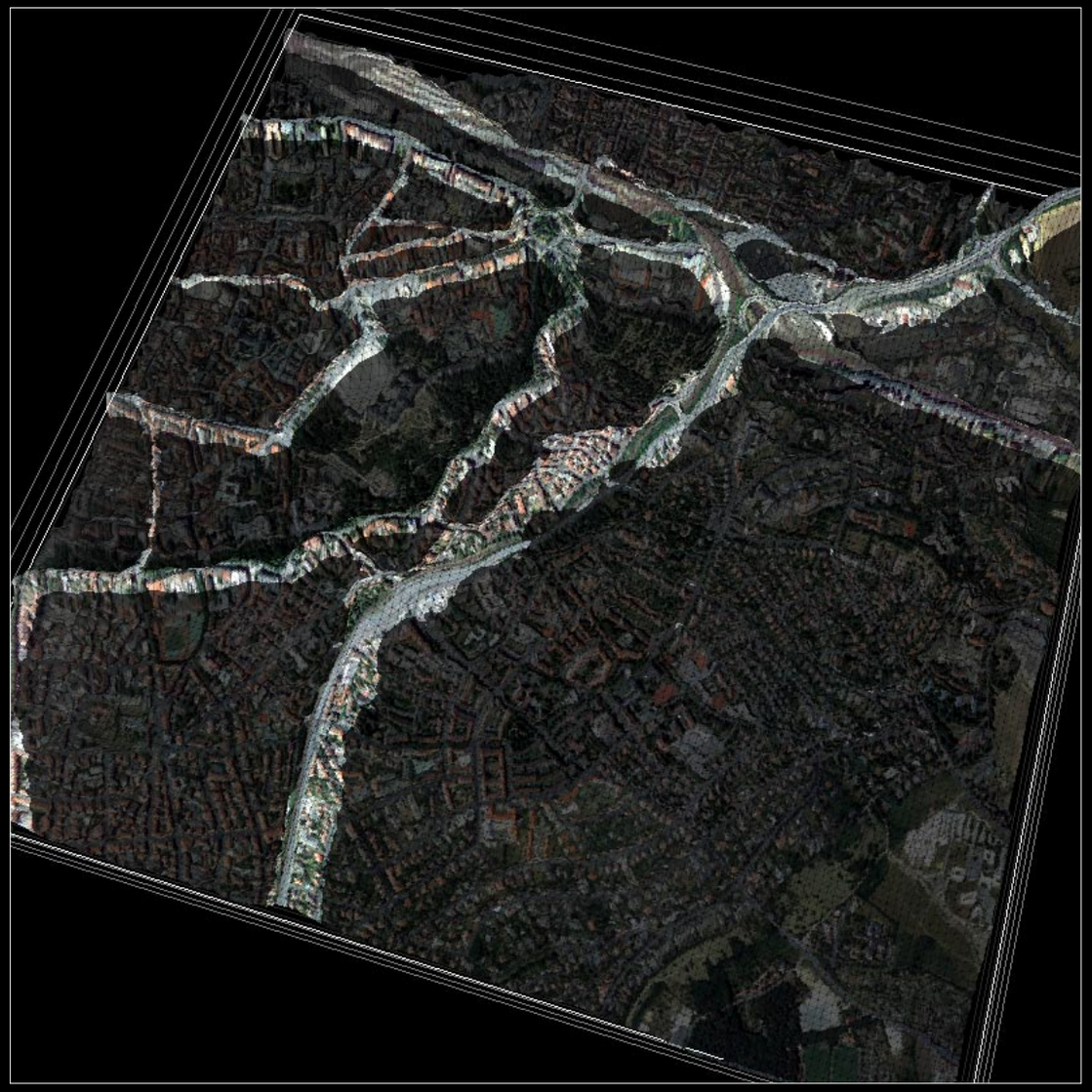
Yann Françoise hat in einem Pilotprojekt eine bereits sehr umfangreiche, interaktive Lärmkarte des Pariser Stadtgebiets dargestellt. Die Karten sind im Internet zwei- und dreidimensional einsehbar. Durch spezielle Filter kann z.B. zwischen Tag und Nacht unterschieden werden.

Auch die Politik wird sich der zunehmenden Bedeutung von Lärm bewusst. Die EU fordert ab Mitte 2007 von ihren Mitgliedsstaaten Lärmkarten für alle Ballungsgebiete mit mehr als 250.000 Einwohnern.

LÄRMKARTE VON WÜRZBURG

Durch eine Satellitenaufnahme und ein Lärmgutachten von 1995, das um Simulationsdaten ergänzt wurde, konnte eine Lärmkarte des Würzburger Stadtgebietes erstellt werden. Beim Einsatz von Filterebenen bleiben besonders die Hauptverkehrsadern sichtbar.



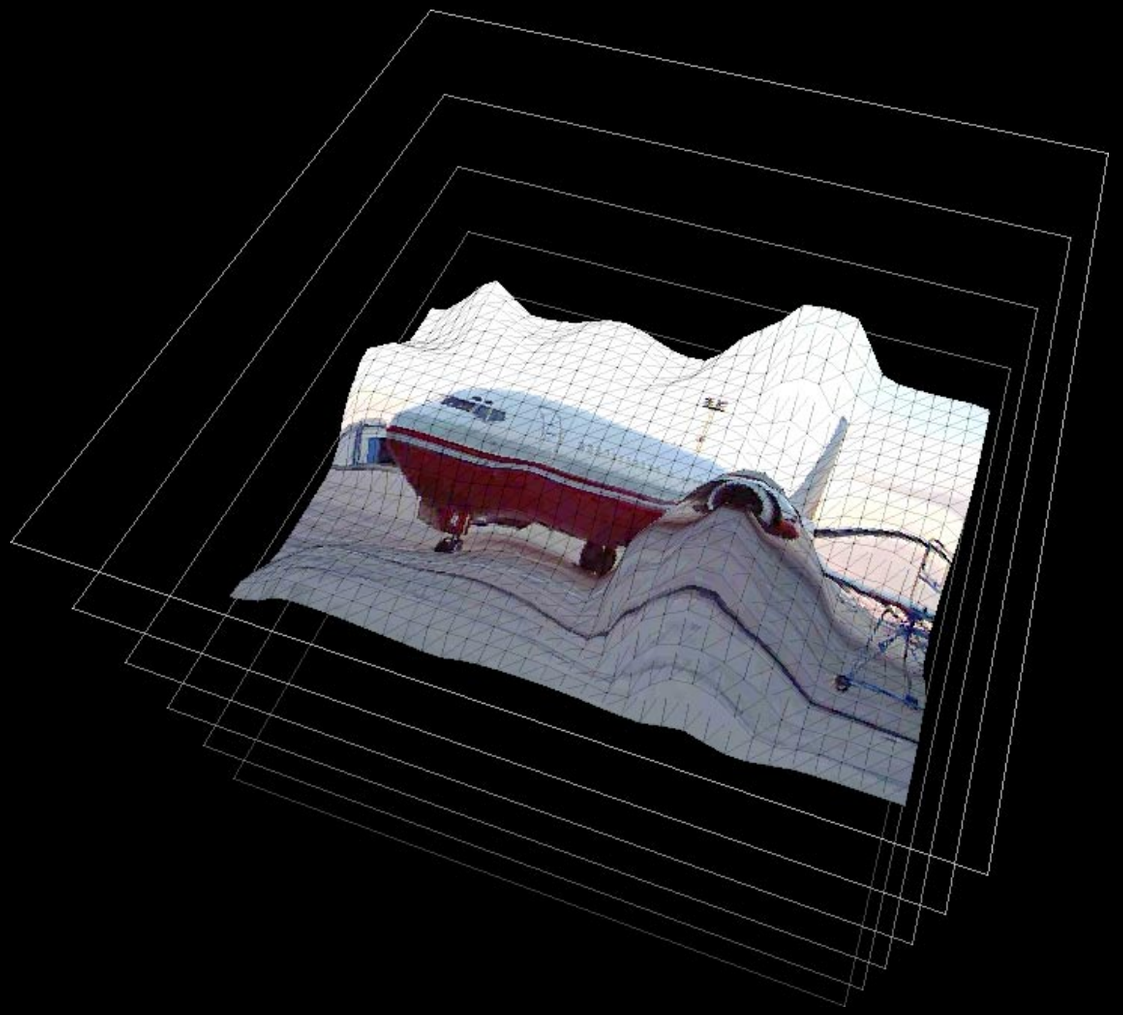


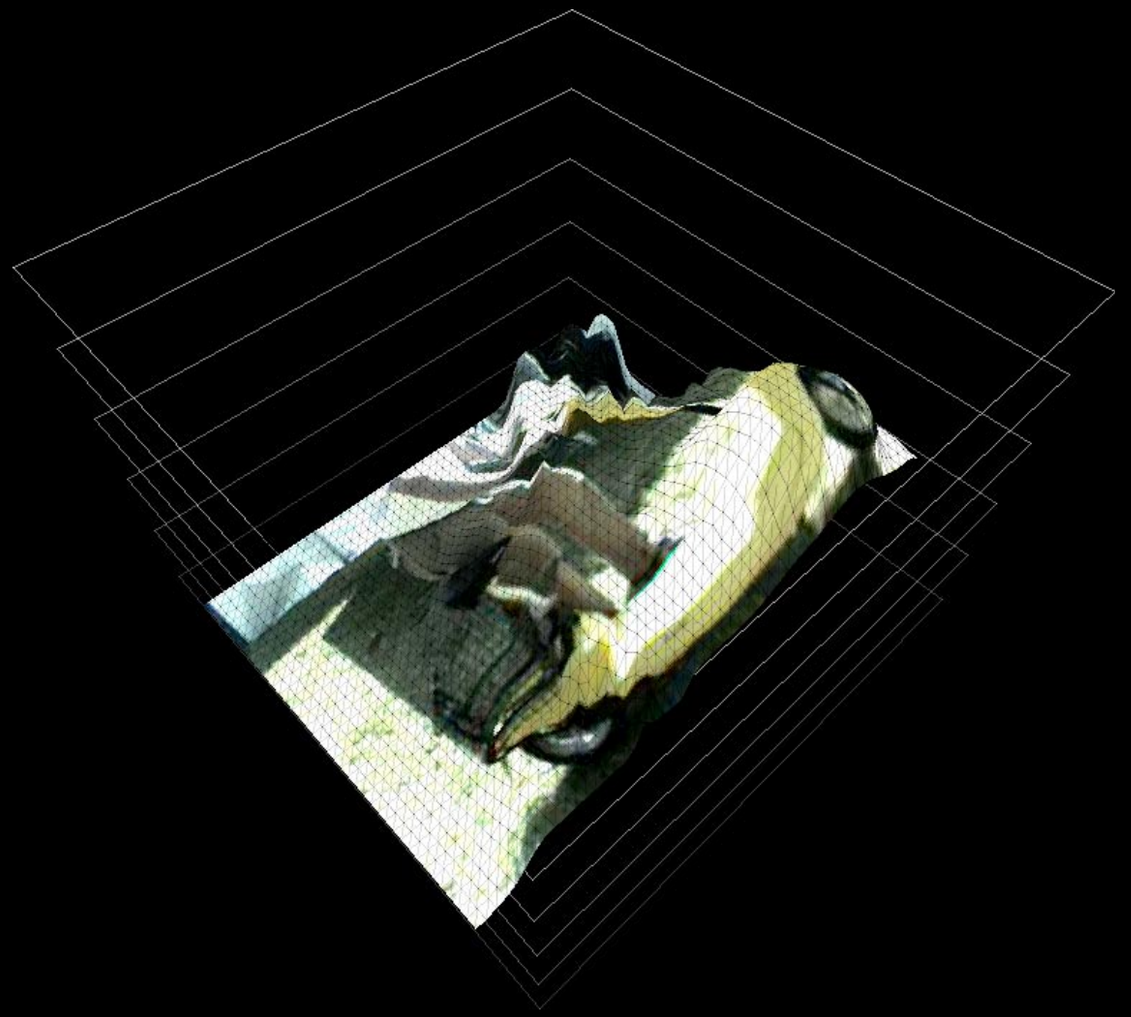
VISUALISIERUNG: WEITERE BEISPIELE

(01 - 05 Original-Aufzeichnungen Dr. G. Heinz, www.acoustic-camera.com)

01 BOEING 737-400

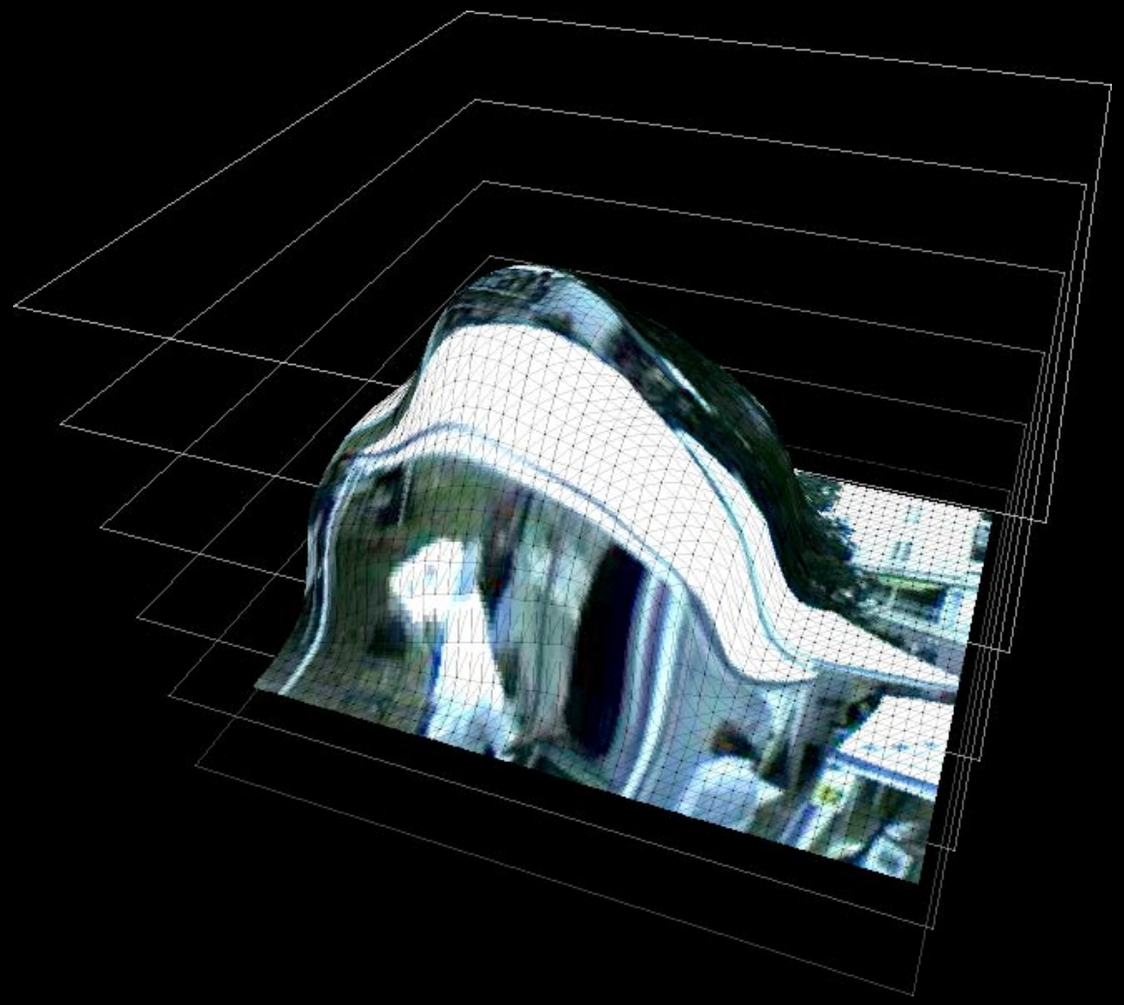
02 ELEKTROROLLER

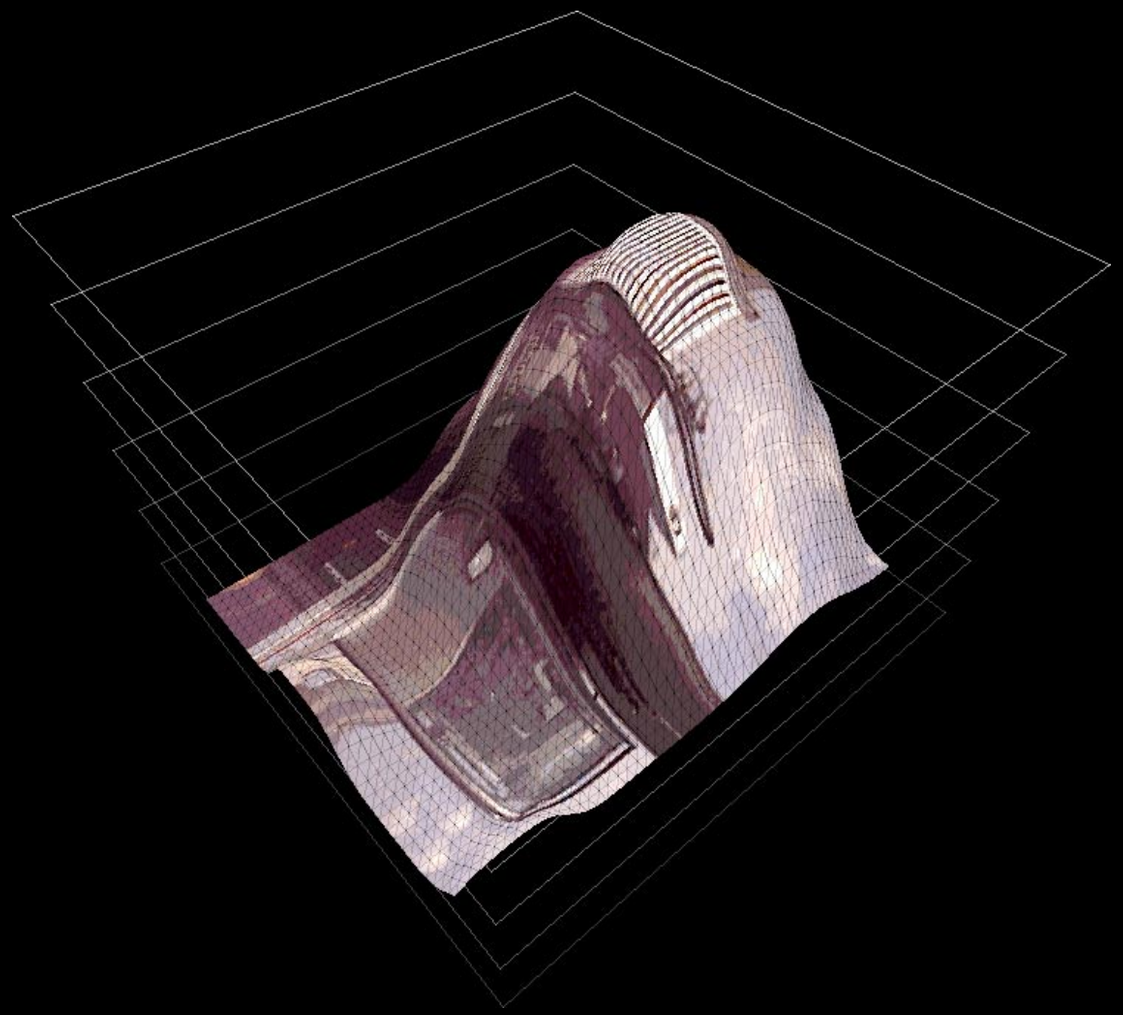


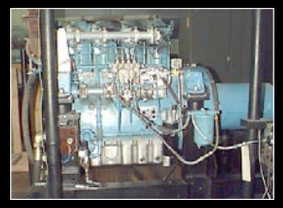


03 OPEN-AIR VERANSTALTUNG

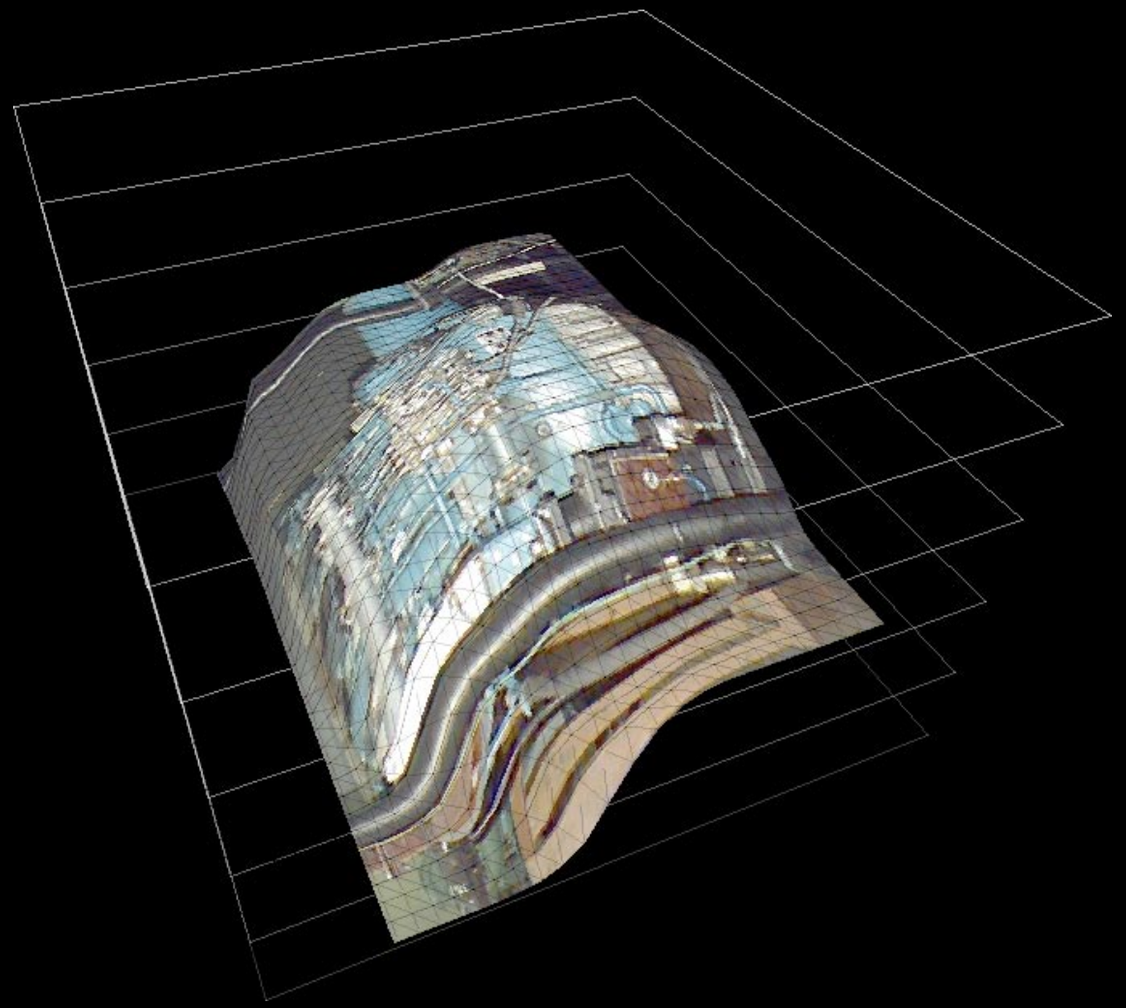
04 BAGGERMOTOR







05 DIESELMOTOR



06 ECHTZEIT-SIMULATION MIT WEBCAM

QUELLEN

QUELLENNACHWEIS

John Maeda, Maeda@Media
Birkhäuser November 2000
ISBN 3925560998

John Maeda, Creative Code
Birkhäuser Oktober 2004
ISBN 3764371080

John Maeda, Design by Numbers
MIT Press October 2001
ISBN 0262632446

Steven Johnson, Interface Culture
Klett-Cotta Juli 1999
ISBN 3608919805

Flash Math Creativity
APress July 2003
ISBN 1590591852

Edward Tufte, The Visual Display Of Quantitative
Information Graphics Press May 2001
ISBN 0961392142

Jef Raskin, Das Intelligente Interface
Addison-Wesley April 2001
ISBN 3827317967

Richard Saul Wurman, Information Anxiety
Pearson Education December 2000
ISBN 0789724103

Frank Hartmann Erwin Bauer, Bildersprache
Facultas Verlag Wien 2002
ISBN 3851147049

Matt Woolman, Digital Information Graphics
Watson-Guptill Publications August 2002
ISBN 0823013537

Matt Woolman, Sonic Graphics
Verlag Schmidt, Mainz 2000
ISBN 3874395405

Nathan Shedroff, Experience Design
New Riders Publishing April 2001
ISBN 0735710783

Peter Wildbur, Michael Burke, Information Graphics
Verlag Schmidt, Mainz Dezember 1998
ISBN 3874394611

Robert Fawcett-Tang, William Owen, Mapping
RotoVision October 2002
ISBN 2880467071

Robert Jacobson, Information Design
MIT Press September 2001
ISBN 0262600358

Design Zentrum München, Der Klang der Dinge
Verlag Silke Schreiber 1993
ISBN 3889600271

Ben Fry, Organic Information Design
Massachusetts Institute of Technology May 2000
http://acg.media.mit.edu/people/fry/thesis/

Ben Fry, Genomic Cartography
Massachusetts Institute of Technology May 2002
http://acg.media.mit.edu/people/fry/

KONTAKT

Daniel Rothaug

daniel@acoustic-cartography.com

http://www.acoustic-cartography.com

http://www.zumkuckuck.com

Fachhochschule Würzburg-Schweinfurt

Fachbereich Gestaltung

http://www.fh-wuerzburg.de/gestaltung/