Proceedings 25. Süddt. Differentialgeometrie-Kolloquium pp. 95–108

DIFFERENTIALGEOMETRIE MIT OPEN GEOMETRY

GEORG GLAESER

ZUSAMMENFASSUNG. OPEN GEOMETRY ist ein geometrisches Programmiersystem, mit dem man Standardprobleme der Differentialgeometrie sehr bequem und komplexere Probleme mit etwas Programmieraufwand lösen und hochqualitativ visualisieren kann. So stehen in der ebenen Kurventheorie alle gängigen Methoden zur Verfügung (Krümmungskreis, Schmiegparabel, adjungierte Kurven wie Evolute, Kaustik, Fußpunktskurven u.v.m.). In der Flächentheorie kann man unmittelbar mit Tangentialebenen, Indikatrizen, Hüllflächen bei Bewegungen usw. arbeiten. Selbst komplexe Bilder werden schattiert auf Standard-PCs „in Echtzeit“ aufgebaut, was interaktive Animationen erlaubt. Einzelbilder lassen sich photorealistisch rendern.

1. EIN EINFÜHRENDES BEISPIEL

Es gibt am Markt mehrere mathematisch-geometrische Programme, die es ermöglichen, recht komplexe geometrische Aufgaben zu lösen und die Ergebnisse zu visualisieren. Als Beispiele seien MATHEMATICA (*www. mathematica. com*), MAPLE (*www. maplesoft. com*), MATHCAD (*www. mathcad. com*), 3D-STUDIO-MAX (*www. 3dmax. com*) genannt. Jedes noch so ausgeklügelte System gelangt jedoch an seine Grenzen, wenn der Benutzer Neuland betreten und nicht mehr Standardprobleme lösen will. Das war der Grund, warum sich die Autoren von OPEN GEOMETRY entschlossen haben, ein „offenes“ System mitsamt dem vollständigen Programmiercode zur Verfügung zu stellen. Es sei jedoch bemerkt, dass OPEN GEOMETRY nicht unbedingt eine „Konkurrenz“ zu all diesen Systemen darstellen soll, sondern eine sinnvolle Ergänzung, die Nischen abdeckt. Man könnte sagen: OPEN GEOMETRY *ist ein Programm für Personen, die sich intensiv mit Geometrie beschäftigen, geschrieben von ebensolchen Personen.*

2000 *Mathematics Subject Classification.* 68U05, 53-04.

 ISBN 3-902233-00-1

Die Vollversion des Programmiersystems läuft am besten unter den Betriebssystemen WINDOWS NT bzw. WINDOWS 9X, die neue Version auch unter LINUX. Nachdem die neue Version in das leistungsfähige und kostenlose Raytracing-Programm POVRAY exportieren kann, empfiehlt es sich, auch dieses Programm zu installieren (*www. povray. org*).

Wenn man über ein Programmiersystem spricht, kommt man nicht umhin zu zeigen, was unter Programmierung zu verstehen ist. OPEN GEOMETRY arbeitet in einer C++-Umgebung und unterstützt demgemäß den gesamten Standard dieser modernen und weitverbreiteten Computersprache. Darüber hinaus stellt es eine Fülle von Klassen und Prozeduren zur Verfügung, die auf geometrische Probleme zugeschnitten sind. Sie erleichtern dem Benutzer — der ja in den seltensten Fällen ein professioneller Programmierer ist — das Erstellen eines lauffähigen Programms wesentlich, indem dieser nicht ständig das Rad neu erfinden muss. Es fängt damit an, dass man nicht wissen muss, wie man ein Fenster am Bildschirm öffnet oder wie man eine Zahl dort ausgibt. OPEN GEOMETRY erledigt diese Aufgaben — unter Benützung der effizienten Geometrie-Bibliothek OPEN GL — stillschweigend im Hintergrund.

Ein OPEN GEOMETRY-Programm hat im Wesentlichen vier für den Benutzer interessante Teile, die in einer Programm-Maske zu finden sind:

```
#include "opengeom.h" // Die Schnittstelle zur Bibliothek
// Hier sollten Sie Ihre globalen Variablen deklarieren
void Scene::Init( )
{
    // Hier sollten Sie Ihre globalen Variablen initialisieren
}
void Scene::Draw( )
{
    // Eigentlicher Zeichenteil.
}
void Scene::Animate( )
{
    // Hier sollten Sie Ihre globalen Variablen "animieren"
    // (drehen, verschieben, skalieren u.v.m.). Dieser Teil
    // ist einer der Staerken von O.G.
}
void Projection::Def( )
{
    // Hier kann die "Kamera" veraendert werden
}
```

Für ein erstes Programm brauchen Sie sich um die Animation bzw. Kamera-Einstellung nicht kümmern, wenn Sie Standard-Werte wählen. Sie inkludieren dann nur eine Datei `defaults2d.h` bzw. `defaults-3d.h`[1].

Wir wollen nun aus der vordefinierten Klasse `ParamCurve2d` eine spezielle Kurve — z.B. eine Astroide — ableiten, diese zeichnen, einen Punkt samt Tangente, Kurvennormale und Krümmungskreis eintragen, sowie die Evolute der Kurve darstellen und den Berührpunkt mit der Kurvennormalen markieren (Figur 1 links oben).

```
// Programm zum Zeichnen einer Kurve samt Evolute.
// Punkt+Tangente, Normale, Kruemmungskreis
#include "opengeom.h"
#include "defaults2d.h"  // typisches 2D-Fenster

// Wir definieren eine neue Kurve, die wir von
// der Open Geometry-Klasse ParamCurve2d "ableiten"
class SomeCurve: public ParamCurve2d
{
public:
    P2d CurvePoint( Real u )
    { // Astroide
        const Real a = 3;
        Real x, y;
        x = a * pow( cos( u ), 3 );  y = a * pow( sin( u ), 3 );
        return P2d( x, y );
    }
};

SomeCurve Curve;
L2d Evolute;

void Scene::Init( )
{
    int n = 300; // so viele Punkte auf der Kurve
    Curve.Def( Black, n, -PI, PI ); // Parametergrenzen
    Evolute.Def( Green, n );
    Curve.GetEvolute( -PI, PI, Evolute );
}

void Scene::Draw( )
{
```

[1]Das Programmpaket beinhaltet einige hundert Demo-Programme (die teilweise in [1], vollständig aber in [2] dokumentiert sind), welche die verschiedensten Aufgaben lösen. Der Benutzer wird sich das am besten passende Programm auswählen und dann nach eigenen Wünschen abändern. Dies erspart eine Menge Schreibarbeit.

```
    ShowAxes( Black, 4, 4 );
    // Tangente im Punkt zum Parameter u0
    Real u0 = 0.7;
    Curve.Tangent( u0 ).Draw( Gray, -3, 3, THIN );
    // Kruemmungskreis im Punkt zum Parameter u0
    Circ2d osc_circ;
    Curve.GetOsculatingCircle( u0, Gray, osc_circ );
    osc_circ.Draw( THIN );
    P2d M = osc_circ.Mid( );
    // Normale im Punkt zum Parameter u0
    P2d P = Curve.CurvePoint( u0 );
    StrL2d n = Curve.Normal( u0 );
    n.Draw( Gray, 0, -2 * M.Distance( P ), THIN );
    // Kurven zeichnen
    Evolute.Draw( MEDIUM, 5 );
    Curve.Draw( THICK, 5 );
    // Punkte markieren
    M.Mark( Green, 0.15, 0.08 );
    P.Mark( Red, 0.15, 0.1 );
}
```

Statt der Methode `GetEvolute( )` können Sie zahlreiche andere Methoden wählen, etwa:

```
GetCata( )          // Katakaustik
GetPedal( )         // Fusspunktskurve
GetAntiPedal( )     // Anti-Fusspunktskurve
GetOrtho( )         // Gegenpunktskurve
GetAntiOrtho( )     // Anti-Gegenpunktskurve
GetOffset( )        // Parallelkurve
GetInvolute( )      // Evolvente
```

Damit lassen sich mit dem aufgelisteten Programm alle Kurven, die in Figur 1 zu sehen, erzeugen. Sämtliche Methoden sind entsprechend dokumentiert und mit Beipielen illustriert ([2]). In weiterer Folge kann man die neu erhaltenen Kurven natürlich manipulieren (verschieben, drehen, skalieren u.v.m.), aber z.B. auch mit Geraden schneiden, und natürlich „animieren“, indem wir die Angabeelemente variieren.

2. Bahnkurven

Eine wichtige Kurvenart ist die flexible Klasse[2] `PathCurve2d`. Jedesmal, wenn wir einen Punkt hinzufügen wollen, verwenden wir die zugehörige Methode `AddPoint( )`. Damit können wir beliebige Bahnkurven erfassen und weiterverarbeiten.

[2]Für Nicht-Programmierer: Das Wort *Klasse* wird in der objekt-orientierten Programmierung in einem viel weiteren Sinn als in der Differentiageometrie verwendet.

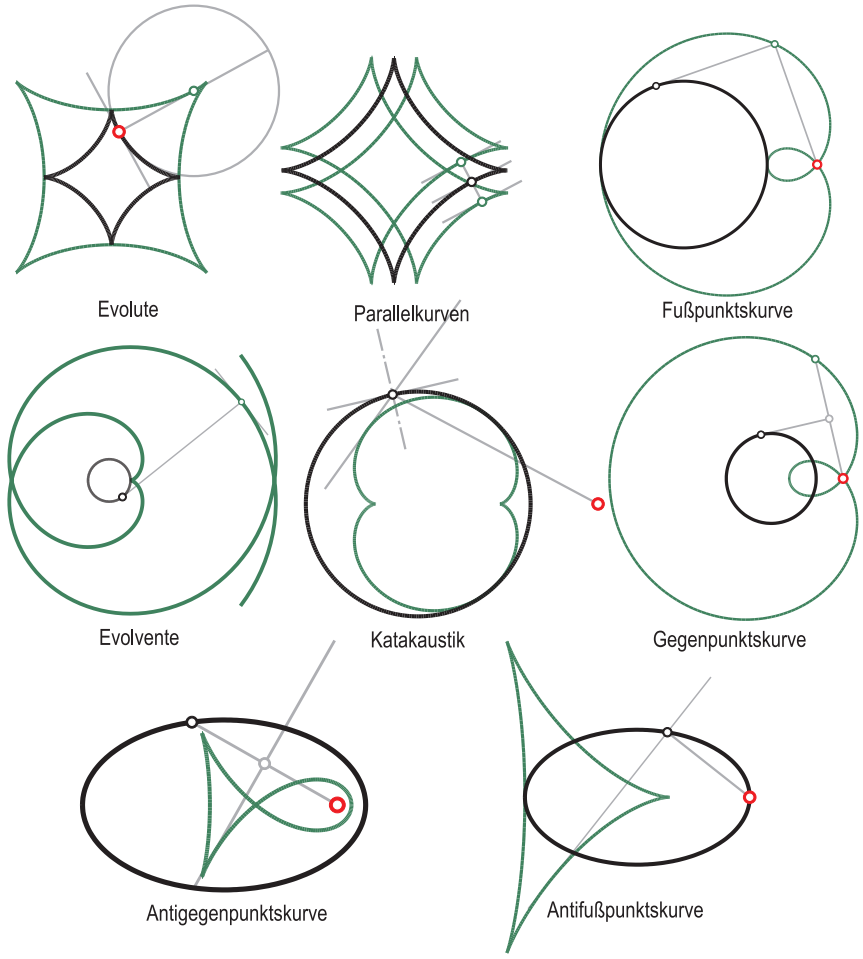


Abbildung 1. Assoziierte Kurven

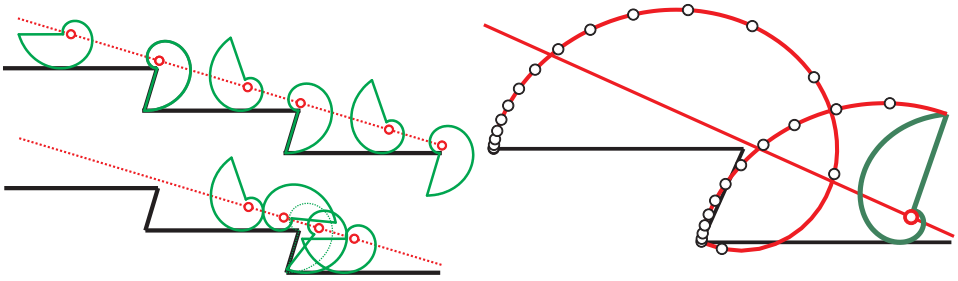
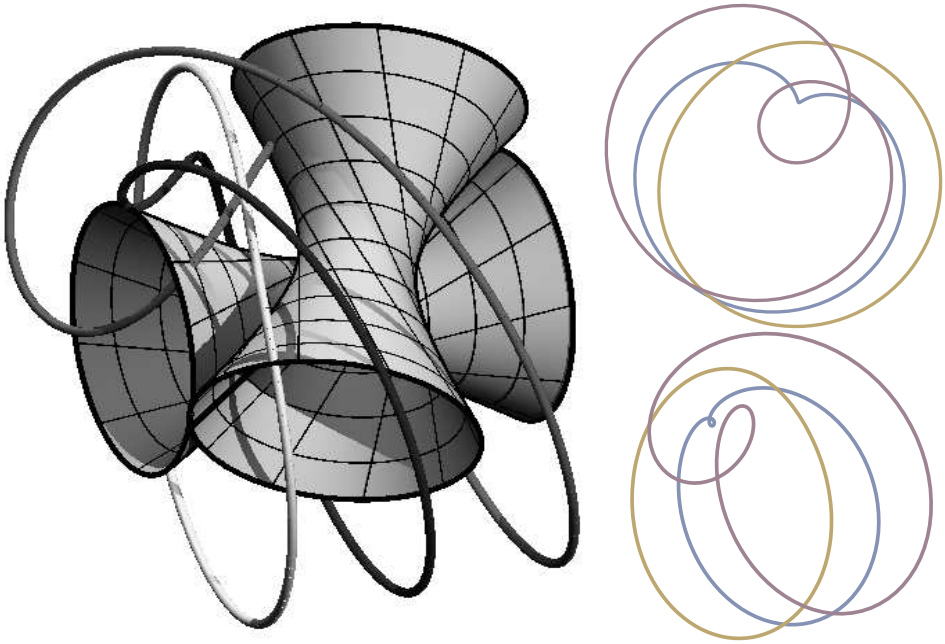


Abbildung 2. Rollende logarithmische Spirale

Als Beispiel diene die Analyse der „rollenden Spirale“: Wenn eine logarithmische Spirale auf entsprechend gebauten Stufen rollt, bewegt sich ihr Zentrum auf einer Geraden (Figur 2). Bei der Anlayse mittels

`PathCurve2d` erkennt man aber zwei Dinge auf Anhieb: Erstens ist diese Rollung in der Praxis kaum brauchbar, weil die Beschleunigungsverhältnisse ungünstig sind, und zweitens kommt es ab einem gewissen Kurswinkel der Spirale zu Unterschnitten (Figur 2 rechts). Diesen Aspekt der Graphikprogrammierung sollte man nicht unterschätzen: *Allein durch Variation der Angabeelemente erkennt man oft zwanglos Dinge, an die man zunächst u.U. gar nicht gedacht hätte.*

Die Bahnkurven sind natürlich nicht nur in der Differentialgeometrie nützlich, sondern sie sind ein entscheidendes Hilfsmittel in jeder Art von „animierter Geometrie“, insbesondere in der Kinematik.



Links: Abb. 3. Nicht parameterisierte Bahnkurven
Rechts: Abb. 4. Ansichten

Die in der Praxis sehr effiziente Methode des „Punkte-Addierens“ ist auch bei der entsprechenden Klasse `PathCurve3d` implementiert. So sind etwa in Figur 3 die Bahnkurven von drei Punkten bei der „Aufeinanderschrotung“ zweier Drehhyperboloide eingezeichnet, ohne dass deren Parameterdarstellung bekannt wäre. Die Bahnkurven (verallgemeinerte Rotoiden) erscheinen in den Hauptrissen (vgl. Grundriss in Figur 4 unten) recht komplex (höhere Radlinien). Bei Parallelprojektion in Richtung der Achse des bewegten Hyperboloids lässt sich jedoch vermuten, dass die Kurven Pascal-Schnecken sind (Figur 4 oben). Dies kann *nachträglich* durch Rechnung bestätigt werden. Open Geometry eignet sich durch seinen raschen Bildaufbau und seine interaktive

Bedienung besonders für das heuristische Auffinden solcher spezieller Ansichten.

Prinzipiell macht es in OPEN GEOMETRY kaum einen Unterschied, ob wir im Raum oder in der Ebene arbeiten. Wenn es zu einem räumlichen Objekt oder Vorgang etwas Entsprechendes in der Ebene gibt, dann unterscheiden sich die Klassen nur durch den Zusatz `2d` bzw. `3d`. So gibt es etwa Splinekurven im Raum und in der Ebene, mit denen man Ergebnisse glätten oder ausgleichen kann, wenn man nicht zuviele Zwischenlagen berechnen will.

3. FLÄCHEN

Flächen (mathematische und empirisch erfasste) bzw. Methoden, diese zu bearbeiten, darzustellen usw. sind ein wesentlicher Bestandteil der Programmbibliothek von OPEN GEOMETRY.

Parameterisierte Flächen werden analog zu den parameterisierten Kurven der Ebene aus einer Klasse `ParamSurface` abgeleitet. So definiert man einen Torus wie folgt:

```
class MySurface: public ParamSurface
{
public:
    virtual P3d SurfacePoint( Real u, Real v )
    {
        const Real a = 9, b = 6;
        Real r = a + b * cos( v );
        Real x, y, z;
        x = r * cos( u );  y = r * sin( u );  z = b * sin ( v );
        return P3d( x, y, z );
    }
};
```

Auf dieselbe Art wurden die Superzyklide bzw. die Römerfläche in Figur 5 erzeugt. Im speziellen Fall des Torus kann man auch mit dem Rohrflächentypus `TubularSurface` arbeiten (so wie dies in Figur 6 rechts geschehen ist), oder aber man kann den Torus als Bewegflächen implementieren, indem man jene Fläche aufsucht, die bei Bewegung eines Kreises entsteht (so wie in Figur 6 links).
Will man für Publikationen auch Schlagschatten miteinbeziehen, kann man die Flächen auf Knopfdruck in das Programm POVRAY exportieren. Manchmal wirken Flächen mit einer gewissen Dicke („Wandstärke“) besonders anschaulich (Figur 7) . Man beachte, das OPEN GEOMETRY in der Lage ist, den Umriss einer Fläche explizit zu berechnen bzw. darzustellen.

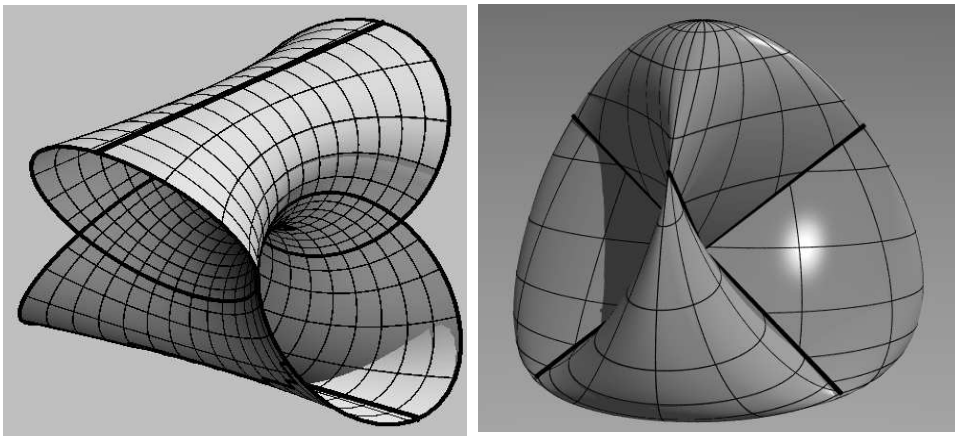


Abbildung 5. Superzyklide und Römerfläche mit O. G.

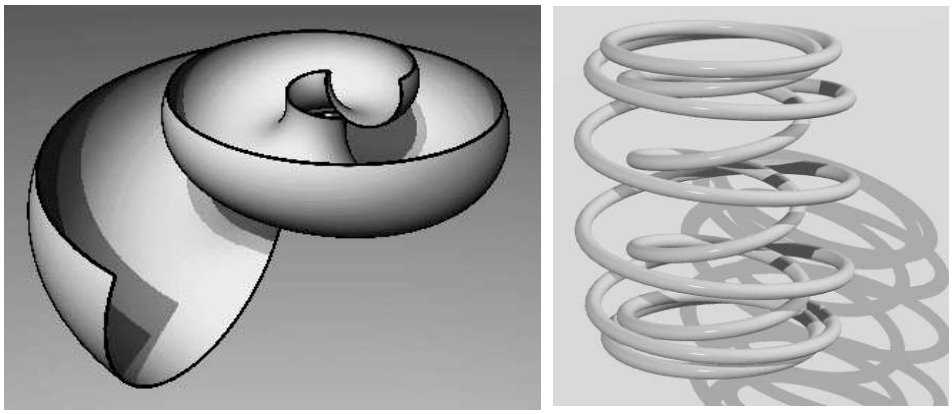


Abbildung 6. Bewegfläche, Rohrfläche

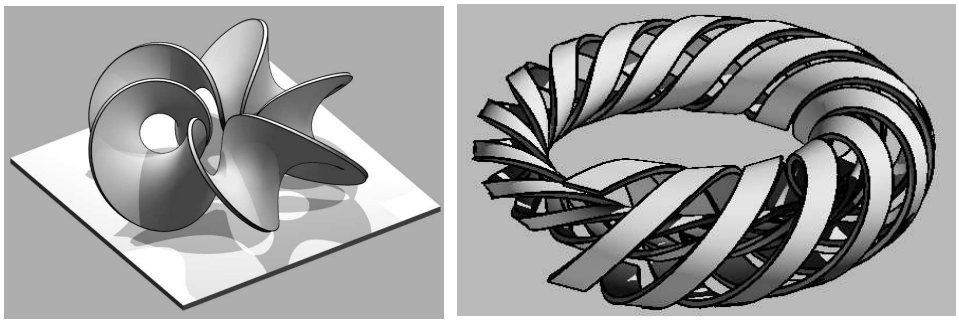


Abbildung 7. Flächen mit „Wandstärke“

Oft erweist es sich als praktisch, eine Fläche durch „sweeping“ erzeugen zu können. Im konkreten Beispiel (Figur 8) wird ein Fräser (eigentlich eine Drehfläche) so längs eines Funktionsgraphen verschoben, dass

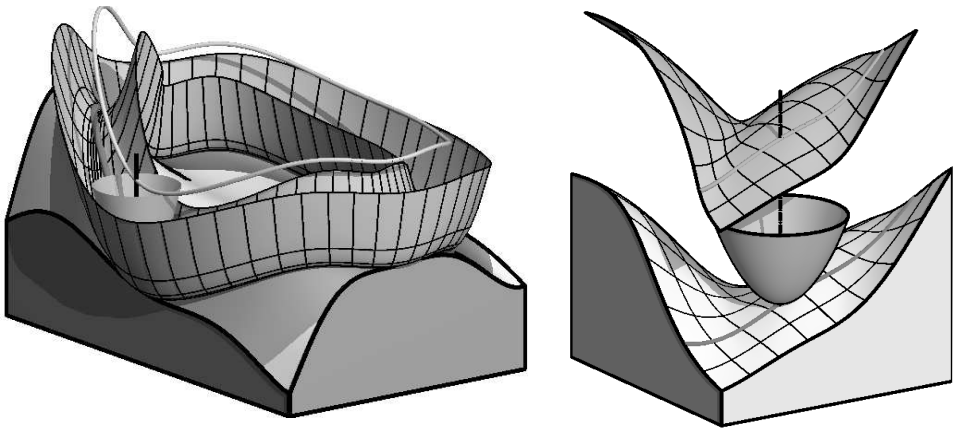


ABBILDUNG 8. Links: „Sweeping“. Rechts: Verallgemeinerte Parallelfläche

die Berührkurve vorgegebene Gestalt hat. Links ist zu sehen, welche Fläche der Fräser einhüllt. Rechts wird jene Fläche veranschaulicht, die ein fester Punkt auf der Fräserachse durchläuft[3].

4. Spezielle Flächenkurven

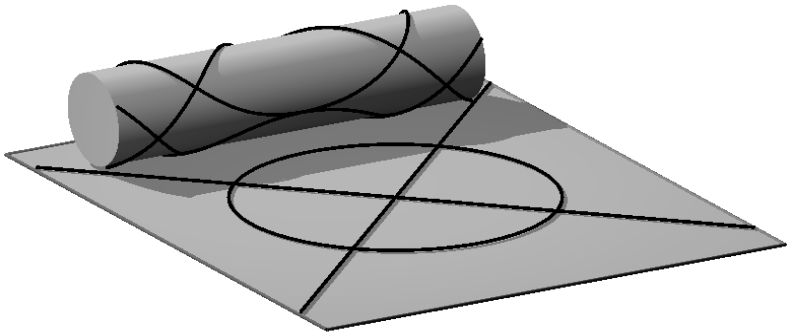


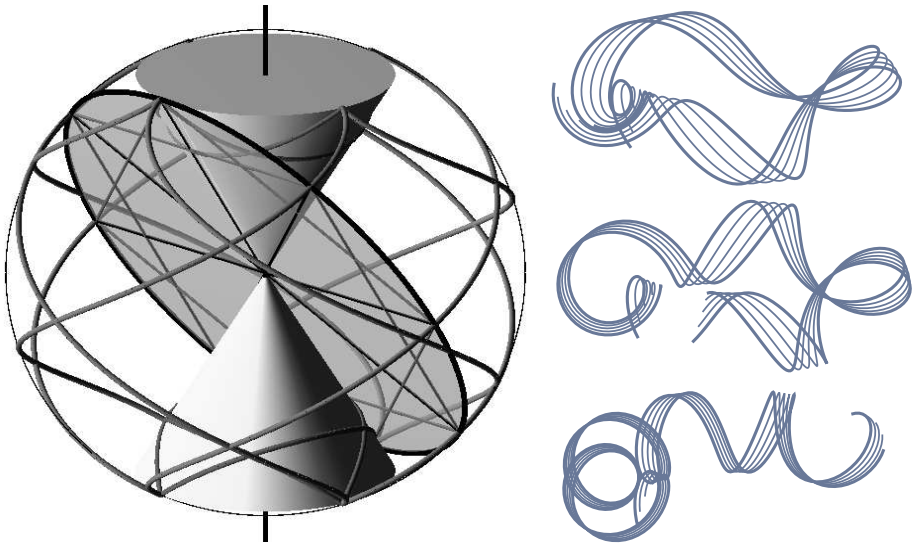
ABBILDUNG 9. Aufwicklung von Kurven auf einen Drehzylinder

In der Differentialgeometrie sind spezielle Flächenkurven von Bedeutung. Neben den (nahezu trivialen) Parameterlinien kann OPEN GEOMETRY beliebige andere Kurven darstellen. So wurden in Figur 9 die „Abdruckspuren“ von zwei Geraden und einem Kreis auf einem Drehzylinder eingezeichnet, wenn dieser über die Basisebene rollt.

[3]H Pottmann, J. Wallner, G. Glaeser: *Collision-free 3-axis milling and selection of cutting-tools*. Computer-Aided Design 31 (1999), 225–232.

Ein ähnlich gelagertes Beispiel (Figur 10) zeigt die Rollung einer Ebene auf einem Drehkegel. Dabei entstehen Böschungslinien auf einer Kugel bzw. als Abdruckspuren u.A. geodätische Linien eines Drehkegels.

In Figur 11 sind Torusloxodromen zu verschiedenen Kurswinkeln zu sehen. Sie entstehen als Lösung von Differentialgleichungen.



Links: Abb. 10. Rollung einer Ebene
Rechts: Abb. 11. Loxodromen am Torus

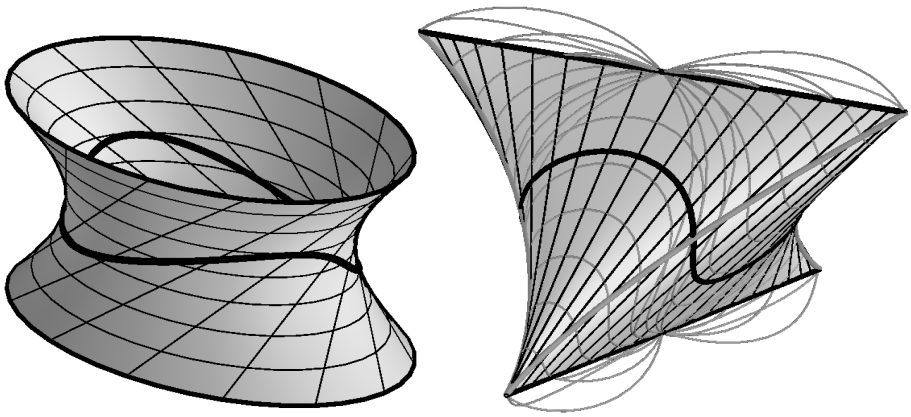


Abbildung 12. Striktionslinien und Fußpunktskurven auf Regelflächen

In Figur 4 sind auf zwei Regelflächen die Striktionslinien eingezeichnet. Im rechten Bild (Reiterfläche) sind weiters die Fußpunktskurven

bezüglich einiger Punkte auf den Leitgeraden der Fläche zu sehen. Der entsprechende Programmiercode ist verblüffend einfach:

```
RiderSurface.DrawPedalLine( Origin, Black, 80, MEDIUM );
RiderSurface.DrawCentralLine( PureRed, 80, MEDIUM );
```

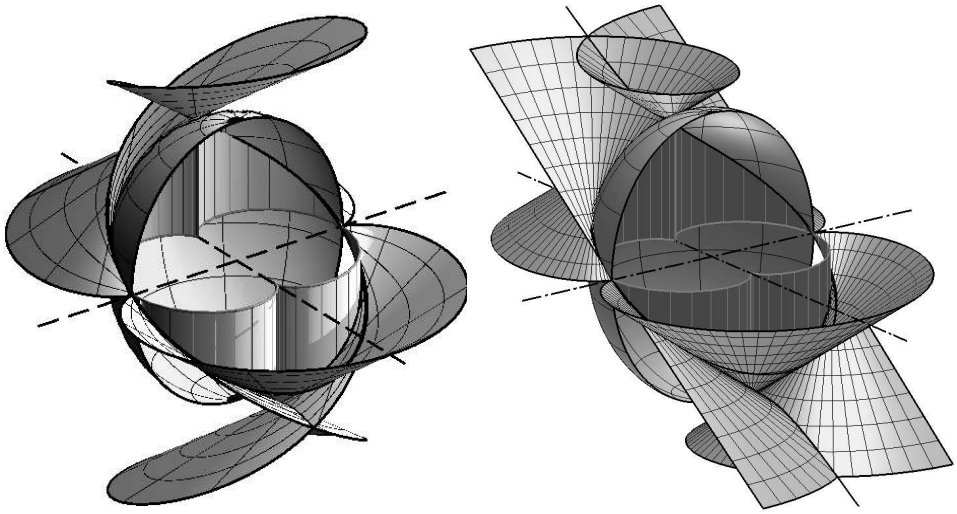


Abbildung 13. Böschungstorse mit verschiedenen Flächenkurven

Als letztes Beispiel diene Figur 13. Im Bild links wurden auf einer von W. Jank[4] untersuchten Böschungstorse die Parameterlinien als Kongruenzschar zur Gratlinie gewählt. Im Bild rechts wurde die Torse durch horizontale Ebenen abgeschnitten. Man beachte auch die Doppelhyperbel, die als Schnittkurve der Fläche mit ihrer Symmetrieebene — also ohne Parameterdarstellung — ermittelt wurde.

5. Geometrische Trickfilme

Open Geometry eignet sich sehr gut zum Herstellen kurzer geometrischer Trickfilme, die durchaus „hochgeometrischen“ Inhalt vermitteln können. Figur 14 veranschaulicht die Abwicklung eines Kegels in die Ebene. Dabei können auch noch verschiedene „Abwickelbewegungen“ vorgegeben werden, Figur 15 illustriert die berühmte Mindingsche Verbiegung des Katenoids in die Wendelfläche.

Trickfilme sind nicht nur didaktisch hilfreich, sie sind manchmal tatsächlich von Nöten, um komplexe Bewegungsvorgänge wirklich zu durchschauen. Die Umwendung des Oloids — wohl eines der schönsten

[4]W. Jank: *Flächen mit kongruenten Fallparabeln* Sb. Österr. Akad. Wiss. **181** (1973), 49–70.

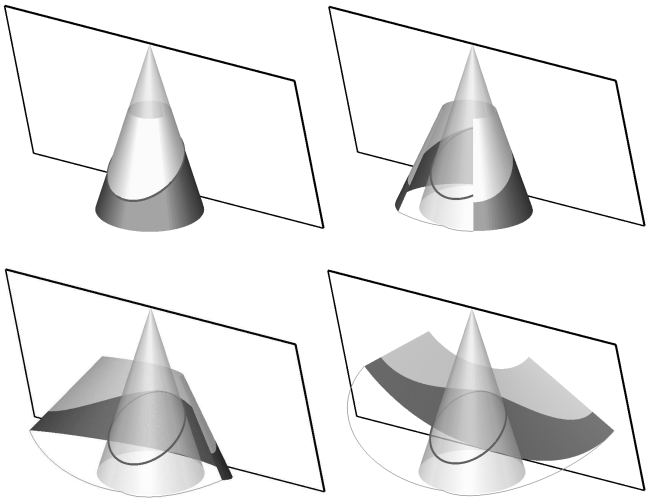


Abbildung 14. Abwicklung eines Kegels

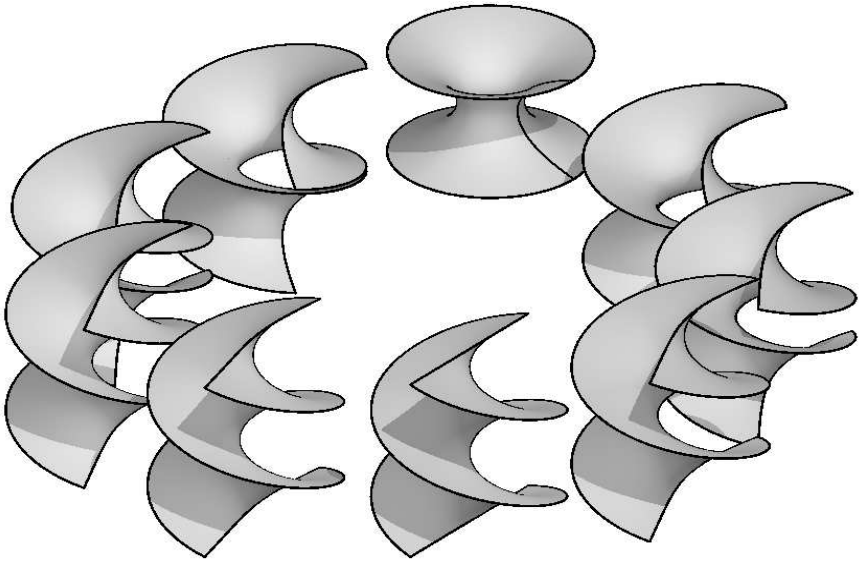


Abbildung 15. Verbiegung des Katenoids in die Wendelfläche

nicht-trivialen Beispiele für eine abwickelbaren Fläche — ist so ein Fall (Figur 16).

Die Bilder werden üblicherweise so schnell erzeugt, dass mit den heutigen PCs und entsprechend schnellen, Open GL hardwaremäßig unterstützenden Graphikkarten Trickfilme in Echtzeit gerechnet werden können. Es besteht aber die Option, kurze Filme als animierte GIF-Dateien anzulegen, wodurch sie auch ins Internet gestellt werden

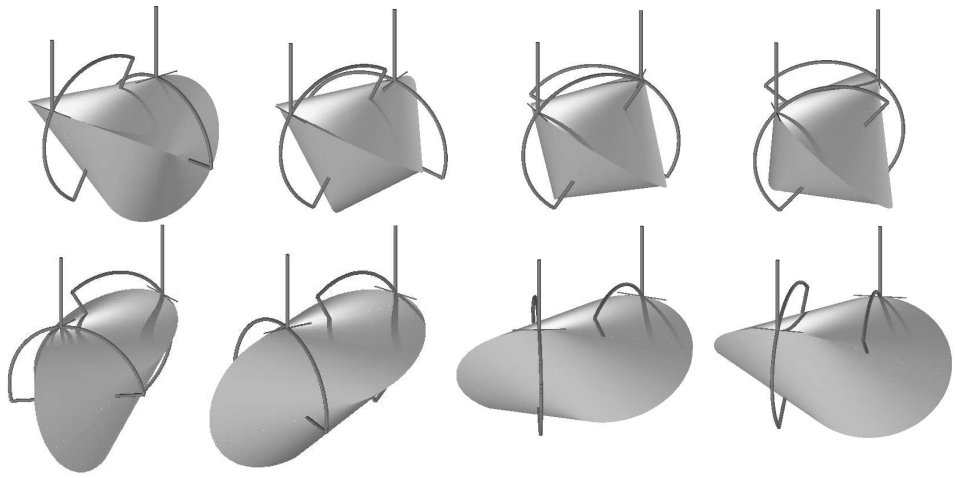


ABBILDUNG 16. Verdrehung des Oloids mittels modifizierter Kardangelenke

können. Der Speicherbedarf ist aber recht beträchtlich. Auf der WEB-Seite *www. uni-ak. ac. at/opengeom* findet man solche Animationen, aber auch direkt ausführbare Demoprogramme.

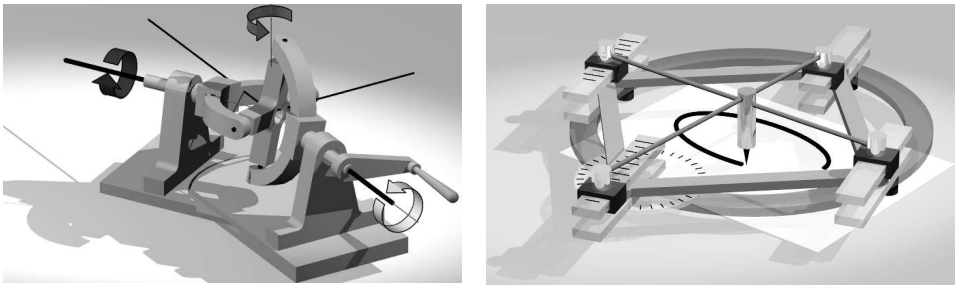


ABBILDUNG 17. Kinematik mit O. G.

Unter anderem gibt es dort auch Anmationen, die nicht unbedingt in den Bereich Differentialgeometrie fallen, sondern eher in das Gebiet der Kinematik (Figur 17).

6. ZUSAMMENFASSUNG

Aus dem bisher Gesagten geht bereits hervor, dass sich OPEN GEOMETRY gut zum Erstellen von Illustrationen für wissenschaftliche Publikationen eignet. Dies war auch einer der Hauptgründe, warum das Programmierpaket überhaupt entwickelt wurde. Im Laufe der Jahre waren die Autoren ständig neuen Aufgaben gegenübergestellt, deren

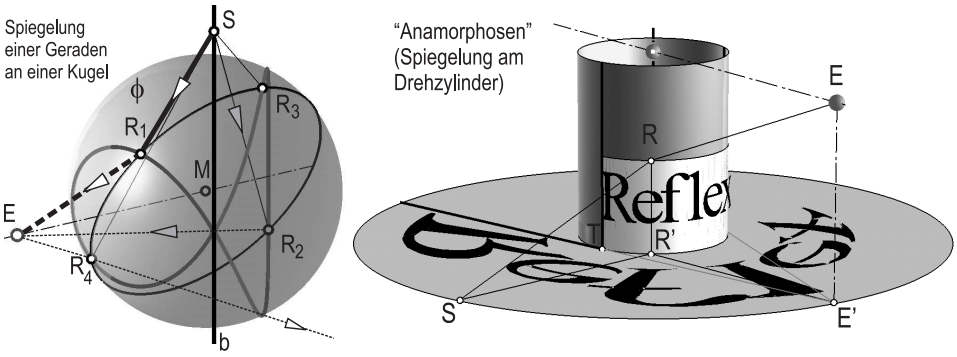


Abbildung 18. Reflexionen mit O. G.

Lösung die Implementierung neuer Module erforderlich machte. So findet man in Figur 18 Illustrationen zur Theorie der Spiegelung an Kugel und Drehzylinder.

Eine weitere Stärke von Open Geometry ist die Möglichkeit, durch Programmieren Animationseffekte zu erreichen, die das Erstellen von Trickfilmen sehr vereinfachen. Wegen der hohen Ausführungsgeschwindigkeit können solche Animationen bei Vorlesungen oder Vorträgen zur Anwendung kommen.

Dokumentation der Software

[1] G. Glaeser, H. Stachel: *Open Geometry — Open GL + Advanced Geometry.* Springer, New York 1998.
[2] G. Glaeser, H.P. Schröcker: *Handbook on Open Geometry 2.0.* Springer, New York 2001.
[3] Aktuelle Informationen zu Open Geometry: *www.uni-ak.ac.at/opengeom*

Georg Glaeser
Univ. f. angewandte Kunst Wien
Oskar Kokoschka-Platz 2
A-1010 Wien
email: georg.glaeser@uni-ak.ac.at