Orbiter User's Guide

Anton Betten

May 19, 2020

Abstract

Orbiter is a program package devoted to the field of Algebraic Combinatorics. Specifically, it is aimed at the problem of classifying combinatorial objects and orbit computations, hence the name. Orbiter provides algorithms for effective handling of finite permutation groups in various actions. This guide is targeted for command line interface usage. Programmers who want to use the Orbiter class library in their own C++ program should consult the programmer's guide.

1 Introduction

Orbiter is a computer algebra system for Algebraic Combinatorics, with an emphasis on the classification of combinatorial objects. The kinds of problems for which Orbiter was designed for lie at the interface between group theoretic computations and combinatorics. Classification and computer search are the main topics for which Orbiter is perhaps stronger than other systems. Orbiter hopes to contribute to the knowledge base of combinatorial structures, and to provide useful tools to investigate structures from various points of view, including their symmetry properties. Orbiter code and Orbiter data structures are optimized for efficiency in terms of memory and execution speed. Orbiter is a library of C++ classes, together with a command line driven front end. There is no graphical user interface. The system offers two modes of use, programming or command line interface. This manual is about the command line interface. Readers who are interested in the Orbiter C++ class library should consult the programmer's guide.

2 Installation

The installation of Orbiter requires the following steps:

(a) Ensure that `git` and the C++ development suite are installed (`gnuc` and `make`). Windows users may have to install `cygwin` (plus the extra packages git, make, gnuc). Macintosh users may have to install the xcode development tools from the appstore (it is free). Linux users may have to install the development packages. Orbiter often produces latex reports. In order to compile these files, make sure you have latex installed

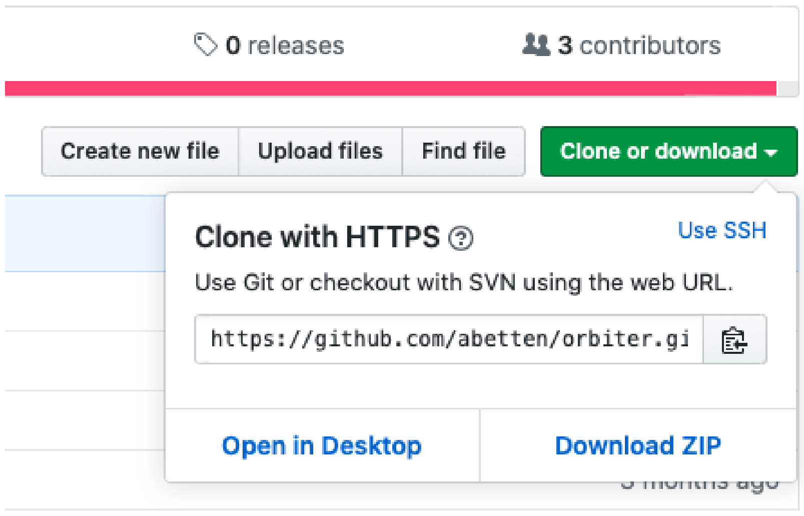


Figure 1: GitHub Clone or Download button

(b) Clone the Orbiter source tree from github (abetten/orbiter). The commands are:

```
git clone <github-orbiter-path>
```

where `<github-orbiter-path>` has to be replaced by the actual address provided by github. To obtain this path, find Orbiter on github, then click on the green box that says “Clone or download” and copy the address into the clipboard by clicking the clipboard symbol (see Figure 1). Back in the terminal, you can paste this text after the `git clone` command.

(c) Issue the following commands to complete the download of submodules:

```
cd orbiter
git submodule init
git submodule update
```

(d) Issue the following commands to compile Orbiter using recursive makefiles:

```
make
make install
```

These two commands compile the Orbiter source tree and copy the executables to the subdirectory bin inside the Orbiter source tree. The orbiter executable is called `orbiter.out`.

3 Finite Fields

Finite fields and projective spaces over finite fields play an important role in Orbiter. The elements of the field $\mathbb{F}_q$ are represented in different ways. Suppose that $q = p^e$ for some prime p and some integer $q \geq 1$. The elements of $\mathbb{F}_q$ are mapped bijectively to the integers in the interval $[0, q-1]$, using the base-p representation. If $e = 1$, the map takes the residue class a mod p with $0 \leq a < p$ to the integer a. Otherwise, we write the field element as

$$\sum_{h=0}^{e-1} a_i \alpha^i$$

where α is the root of some irreducible polynomial $m(X)$ of degree e over $\mathbb{F}_p$ amd $0 \leq a_i < p$ for all i. The associated integer is obtained as

$$\sum_{h=0}^{e-1} a_i p^i.$$

This is the numerical rank of the polynomial. This representation takes 0 in $\mathbb{F}_q$ to the integer 0. Likewise, $1 \in \mathbb{F}_q$ is mapped to the integer 1. Arithmetic is done by considering the polynomials over $\mathbb{F}_p$ and modulo the irreducible polynomial $m(X)$ with root α. For instance, the field $\mathbb{F}_4$ is created using the polynomial $m(X) = X^2 + X + 1$. The elements are

$$0, \quad 1, \quad 2 = \alpha, \quad 3 = \alpha + 1.$$

Orbiter maintains a small database of primitive (irreducible) polynomials for the purposes of creating finite fields. This means that the residue class of α is a primitive element of the field, where α is a root of the polynomial.

The command

```
orbiter.out -cheat_sheet_GF 4
```

creates a report for the field $\mathbb{F}_4$.

Unlike other computer algebra systems (GAP [6] and Magma [5]) Orbiter does not use Conway polynomials. However, Orbiter provides the option to override the polynomial used to create the finite field. For subfield relationships, the cheat sheet for the large field will indicate the irreducible polynmials of the subfield. For instance, Table 1 shows the subfields of $\mathbb{F}_{64}$ generated by the polynomial $X^6 + X^5 + 1$ whose numerical rank is 97.

4 Finite Projective Spaces

Finite projective spaces and their groups are essential objects in Orbiter. The projective space $\mathrm{PG}(n, q)$ is the set of non-zero subspaces of $\mathbb{F}_q^{n+1}$ ordered with respect to inclusion. The projective dimension of a subspace is always one less than the vector space dimension.

Subfield	Polynomial	Numerical rank
$\mathbb{F}_4$	X^2+X+1	7
$\mathbb{F}_8$	X^3+X+1	11

Table 1: The subfields of $\mathbb{F}_{64}$

So, a projective point is a vector subspace of dimension one. A projective line is a vector subspace of dimension two, etc. A point is written as $P(\mathbf{x})$ for some vector $\mathbf{x} = (x_0, \ldots, x_n)$ with $x_0, \ldots, x_n \in \mathbb{F}_q$, not all zero. For any non-zero element $\lambda \in \mathbb{F}_q$, $P(\lambda\mathbf{x})$ is the same point as $P(\mathbf{x})$. For $\mathbf{a} = (a_0, \ldots, a_n) \in \mathbb{F}_q$, not all zero, the symbol $[\mathbf{a}]$ represents the line

$$\{P(\mathbf{x}) \mid \mathbf{a} \cdot \mathbf{x} = \sum_{i=0}^{n} a_i x_i = 0\}.$$

For any non-zero element $\lambda \in \mathbb{F}_q$, $[\lambda\mathbf{a}] = [\mathbf{a}]$.

The command

```
orbiter.out -cheat_sheet_PG 3 2
```

creates a report for the projective geometry $\mathrm{PG}(3,2)$. Orbiter has enumerators for points and subspaces of $\mathrm{PG}(n,q)$. The point enumerator allows to represent the points using the integer interval $[0, \theta_n(q) - 1]$, where

$$\theta_n(q) = \frac{q^{n+1}-1}{q-1}.$$

The points in projective geometry are the one-dimensional subspaces.

The permutation representation of various groups acting on projective spave is based on an enumeration of the elements of $\mathfrak{G}r_1$. By enumerating $\mathfrak{G}r_1$ we mean that we choose a fixed bijection between the set $\mathfrak{G}r_1$ and the integer interval $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$ where $N = \theta_n(q)$. In order to facilitate the bijection, we enumerate generating vectors. The conditions on the vectors are summarized below:

1. The vector is not the zero vector.
2. The rightmost nonzero entry in the vector is one. If it is not, we nomalize the vector so that the rightmost nonzero vector is indeed one. This operation does not change the projective point which is associated with the vector.

The second condition ensures that we list each projective point exactly once. We require two functions, Rank and Unrank. The function Rank takes a vector $\mathbf{x} \in \mathbb{F}_q^n$, not zero, and maps it to the element in $\mathbb{Z}_N$ representing the projective point $\mathbf{P}(\mathbf{x})$. A frame in $\mathrm{PG}(n,q)$ is a set of $n+2$ points, no $n+1$ in a hyperplane. We assume that the coordinates of a vector are indexed by the elements of $\mathbb{Z}_n$. Also, we let $\mathbf{e}_i$ be the i-th unit vector. A frame for $\mathrm{PG}(n,q)$ is

$$\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}, \mathbf{e}_0 + \cdots + \mathbf{e}_{n-1}.$$

This is the *standard frame.* We start the labeling of points with the standard frame. After these $n+2$ points, we list the remaining points in lexicographic ordering (of the right normalized representative). Thus, for $\mathrm{PG}(2,2)$ the ordering is

$$(1,0,0),\ (0,1,0),\ (0,0,1),\ (1,1,1),\ (1,1,0),\ (1,0,1),\ (0,1,1).$$

Let us describe the two functions rank and unrank to perform the actual mappings between $\mathrm{PG}(n,q)$ and $\mathbb{Z}_N$, where $N = \theta_n(q)$. For this, assume that ranking and unranking functions have already been defined for the elements of the finite field $\mathbb{F}_q$. Thus, we assume that for $x \in \mathbb{F}_q$, $\textsc{Rank}(\mathbb{F}_q, x)$ is a number b in $\mathbb{Z}_q$. Also, for $b \in \mathbb{Z}_q$, we assume that $\textsc{Unrank}(\mathbb{F}_q, b)$ is the corresponding $x \in \mathbb{F}_q$. So, we assume that Rank and Unrank are mutually inverse functions. Consider the group $\mathrm{PGL}(3,2)$ acting on $\mathrm{PG}(2,2)$, for instance. The points of

Algorithm 1 Rank

```
1: procedure Rank(vector : $\mathbf{x}$, field : $\mathbb{F}_q$, int : $n$)
2:     assert $\mathbf{x}$ is a nonzero vector in $\mathbb{F}_q^n$.
3:     if $\mathbf{x} = \mathbf{e}_i$ then
4:         return $i$
5:     if $\mathbf{x} =$ one then
6:         return $n$
7:     $i \leftarrow \max\{j \in \mathbb{Z}_n \mid x_j \neq 0\}$
8:     $\mathbf{x} \leftarrow \frac{1}{x_i}\mathbf{x}$
9:     $a := 0$
10:    for $j = i-1, \ldots, 1, 0$ do
11:        $a \leftarrow a+$Rank$(\mathbb{F}_q, x_j)$
12:        if $j > 0$ then
13:            $a \leftarrow a \cdot q$
14:    if $i = n-1$ and $a \geq \sum_{j=0}^{i-1} q^j$ then
15:        $a \leftarrow a - 1$
16:    $a \leftarrow a + n - i + \sum_{j=0}^{i-1} q^j$
17:    return $a$
```

$\mathrm{PG}(2,2)$ are listed in 2.

Let V be a finite dimensional vector space and let $\mathfrak{Gr}_k(V)$ be the Grassmannian of k-dimensional subspaces of V. If $\dim(V) = n$, the notation $\mathfrak{Gr}_{n,k}$ is used for $\mathfrak{Gr}_k(V)$. The size of the set $\mathfrak{Gr}_{n,k}$ can be computed as

$$\begin{bmatrix} n \\ k \end{bmatrix}_q = \prod_{i=0}^{k-1} \frac{q^{n-i}-1}{q^{k-i}-1},$$

using the q-binomial coefficient. A system of representatives of the k-dimensional subspaces is obtained from the $k \times n$ RREF (row-reduced echelon form) matrices of rank k. The

Algorithm 2 Unrank

```
procedure Unrank(int : $a$, field : $\mathbb{F}_q$, int : $n$)
    assert $a \in \mathbb{Z}_N$ where $N = \theta_{n-1}(q)$.
    if $a < n$ then
        return $\mathbf{e}_a$
    $a \leftarrow a - n$
    if $a = 0$ then
        return one
    $a \leftarrow a - 1$
    $\mathbf{x} \leftarrow \mathbf{0}$
    for $i = 1, \ldots, n-1$ do
        if $a \geq \sum_{j=1}^{i-1} q^j$ then
            $a \leftarrow a - \sum_{j=1}^{i-1} q^j$
        else
            $x_i \leftarrow 1$
            break
    for $k = i+1, \ldots, n-1$ do
        $x_k \leftarrow 0$
    $a \leftarrow a + 1$
    if $i = n-1$ and $a \geq \sum_{j=0}^{i-1} q^j$ then
        $a \leftarrow a + 1$
    $j \leftarrow 0$
    while $a > 0$ do
        $r \leftarrow a \mod q$
        $x_j \leftarrow$ Unrank($\mathbb{F}_q, r$)
        $j \leftarrow j + 1$
        $a \leftarrow (a - r)/q$
    for $h = j, \ldots, i-1$ do
        $x_h \leftarrow 0$
    return $\mathbf{x}$
```

$a = \text{Rank}(x)$	$\mathbf{x} = \text{Unrank}(a)$
0	$(1,0,0)$
1	$(0,1,0)$
2	$(0,0,1)$
3	$(1,1,1)$
4	$(1,1,0)$
5	$(1,0,1)$
6	$(0,1,1)$

Table 2: Representatives of the points of PG$(2,2)$

$L_0 = \begin{bmatrix} 100 \\ 010 \end{bmatrix}$	$L_4 = \begin{bmatrix} 101 \\ 010 \end{bmatrix}$	$L_8 = \begin{bmatrix} 102 \\ 010 \end{bmatrix}$	$L_{12} = \begin{bmatrix} 010 \\ 001 \end{bmatrix}$
$L_1 = \begin{bmatrix} 100 \\ 011 \end{bmatrix}$	$L_5 = \begin{bmatrix} 101 \\ 011 \end{bmatrix}$	$L_9 = \begin{bmatrix} 102 \\ 011 \end{bmatrix}$	
$L_2 = \begin{bmatrix} 100 \\ 012 \end{bmatrix}$	$L_6 = \begin{bmatrix} 101 \\ 012 \end{bmatrix}$	$L_{10} = \begin{bmatrix} 102 \\ 012 \end{bmatrix}$	
$L_3 = \begin{bmatrix} 100 \\ 001 \end{bmatrix}$	$L_7 = \begin{bmatrix} 110 \\ 001 \end{bmatrix}$	$L_{11} = \begin{bmatrix} 120 \\ 001 \end{bmatrix}$	

Table 3: The 13 lines of $\mathrm{PG}(2,3)$

elements of $\mathrm{PG}(n-1,q)$ are the k-dimensional subspaces of $V = \mathbb{F}_q^n$. It is convenient to identify a subspace with a matrix whose rows contain a basis for it. In coding theory, such a matrix is called a generator matrix. For instance, the 13 lines of $\mathrm{PG}(2,3)$ are shown using 2×3 RREF generator matrices as shown in Table 3.

5 Algebraic Sets

A very important notion in projective geometry is that of algebraic sets. A set of points V in $\mathrm{PG}(n,q)$ is algebraic if there is a set of homogeneous polynomials $p_1,\ldots,p_r$ whose roots over $\mathbb{F}_q$ are the given set. In this case, we write $V = \mathbf{v}(p_1,\ldots,p_r)$. The set V is often called the variety of $p_1,\ldots,p_r$.

Conversely, given a set of points V in $\mathrm{PG}(n,q)$, the ideal $I(V)$ is the set of homogeneous polynomials in $\mathbb{F}_q[X_0,\ldots,X_n]$ which vanish on all of V. This set is an ideal in the polynomial ring. In fact, it is a pricipal ideal, meaning that it is generated by one element only. Orbiter has ways to compute the variety of a polynomial ideal and to compute a generator for the ideal of a set.

Interestingly, in $\mathrm{PG}(n,q)$, every set is algebraic of degree at most $(n+1)(q-1)$ [7]. The associated polynomial is unique and known as the algebraic normal form of the set.

Table 4 shows the Orbiter monomial orderings for degrees $2,3$ and 4 in a plane. Suppose we are interested in $\mathbb{F}_{11}$ rational points of the elliptic curve $y^2 = x^3 + x + 3$. We write $x^3 + 3 - y^2 + x = 0$. Homogenizing yields $X^3 + 3Z^3 - Y^2Z + XZ = 0$. Using X_0, X_1, X_2 instead of X, Y, Z yields

$$X_0^3 + 3X_2^3 + 10X_1^2X_2 + X_0X_2^2 = 0.$$

Using the indexing of monomials from Table 4, we record the following pairs (a,i) where a is the coefficient and i is the index of the monomial

$$(1,0),\ (3,2),\ (10,6),\ (1,7).$$

h	monomial	vector
0	X_0^2	$(2,0,0)$
1	X_1^2	$(0,2,0)$
2	X_2^2	$(0,0,2)$
3	X_0X_1	$(1,1,0)$
4	X_0X_2	$(1,0,1)$
5	X_1X_2	$(0,1,1)$

h	monomial	vector
0	X_0^3	$(3,0,0)$
1	X_1^3	$(0,3,0)$
2	X_2^3	$(0,0,3)$
3	$X_0^2X_1$	$(2,1,0)$
4	$X_0^2X_2$	$(2,0,1)$
5	$X_0X_1^2$	$(1,2,0)$
6	$X_1^2X_2$	$(0,2,1)$
7	$X_0X_2^2$	$(1,0,2)$
8	$X_1X_2^2$	$(0,1,2)$
9	$X_0X_1X_2$	$(1,1,1)$

h	monomial	vector
0	X_0^4	$(4,0,0)$
1	X_1^4	$(0,4,0)$
2	X_2^4	$(0,0,4)$
3	$X_0^3X_1$	$(3,1,0)$
4	$X_0^3X_2$	$(3,0,1)$
5	$X_0X_1^3$	$(1,3,0)$
6	$X_1^3X_2$	$(0,3,1)$
7	$X_0X_2^3$	$(1,0,3)$
8	$X_1X_2^3$	$(0,1,3)$
9	$X_0^2X_1^2$	$(2,2,0)$
10	$X_0^2X_2^2$	$(2,0,2)$
11	$X_1^2X_2^2$	$(0,2,2)$
12	$X_0^2X_1X_2$	$(2,1,1)$
13	$X_0X_1^2X_2$	$(1,2,1)$
14	$X_0X_1X_2^2$	$(1,1,2)$

Table 4: The Orbiter ordering of monomials of degree $2, 3$ and 4 in a plane

This is concatenated to the sequence $1, 0, 3, 2, 10, 6, 1, 7$. The Orbiter command

```
orbiter.out -v 2 -create_combinatorial_object -q 11 -n 2 \
    -projective_variety "EC" 3 "1,0,3,2,10,6,1,7"
```

creates the algebraic set associated to the cubic curve $y^2 = x^3 + x + 3$ in PG$(2, 11)$. It turns out that there are exactly 18 points over $\mathbb{F}_{11}$ (cf. Figure 2).

Table 5 shows the Orbiter monomial orderings for degrees 2 and 3 in PG$(3, q)$.

6 Linear Groups

Groups in Orbiter are always permutation groups. This means that the group comes with a set on which it acts, called the permutation domain. One group can have many different actions. Three basic actions are defined for matrix groups: projective, affine, and general linear. In addition, there are direct product action and tensor product actions. Group theoretic algorithms require short orbits to be efficient. In order to create short orbits, Orbiter tags on an extra permutation domain for product and tensor actions. These extra permutation domains can be blended out later by forming a restricted action.

Let $V \simeq \mathbb{F}_q^n$ be a finite dimensional vector space over $\mathbb{F}_q$. A group G can act on V in one of the types listed in Table 6. The elements of finite fields are represented as integers as

h	monomial	vector
0	X_0^2	$(2,0,0,0)$
1	X_1^2	$(0,2,0,0)$
2	X_2^2	$(0,0,2,0)$
3	X_3^2	$(0,0,0,2)$
4	X_0X_1	$(1,1,0,0)$
5	X_0X_2	$(1,0,1,0)$
6	X_0X_3	$(1,0,0,1)$
7	X_1X_2	$(0,1,1,0)$
8	X_1X_3	$(0,1,0,1)$
9	X_2X_3	$(0,0,1,1)$

h	monomial	vector
0	X_0^3	$(3,0,0,0)$
1	X_1^3	$(0,3,0,0)$
2	X_2^3	$(0,0,3,0)$
3	X_3^3	$(0,0,0,3)$
4	$X_0^2X_1$	$(2,1,0,0)$
5	$X_0^2X_2$	$(2,0,1,0)$
6	$X_0^2X_3$	$(2,0,0,1)$
7	$X_0X_1^2$	$(1,2,0,0)$
8	$X_1^2X_2$	$(0,2,1,0)$
9	$X_1^2X_3$	$(0,2,0,1)$
10	$X_0X_2^2$	$(1,0,2,0)$
11	$X_1X_2^2$	$(0,1,2,0)$
12	$X_2^2X_3$	$(0,0,2,1)$
13	$X_0X_3^2$	$(1,0,0,2)$
14	$X_1X_3^2$	$(0,1,0,2)$
15	$X_2X_3^2$	$(0,0,1,2)$
16	$X_0X_1X_2$	$(1,1,1,0)$
17	$X_0X_1X_3$	$(1,1,0,1)$
18	$X_0X_2X_3$	$(1,0,1,1)$
19	$X_1X_2X_3$	$(0,1,1,1)$

Table 5: The Orbiter ordering of monomials of degree 2 and 3 in three-space

Type	Perm. Domain	Degree
General linear $\mathrm{GL}(n,q)$	all vectors of V	q^n
Affine $\mathrm{AGL}(n,q)$	all vectors of V	q^n
Projective $\mathrm{PGL}(n,q)$	$\mathfrak{G}r_1(V)$	$\frac{q^n-1}{q-1}$
Wreath product $\mathrm{GL}(d,q) \wr \mathrm{Sym}(n)$	$\mathfrak{G}r_1((\mathbb{F}_q^d)^{\otimes n})$ extended	$n + nq^d + \frac{q^{d^n}-1}{q-1}$

Table 6: Basic actions

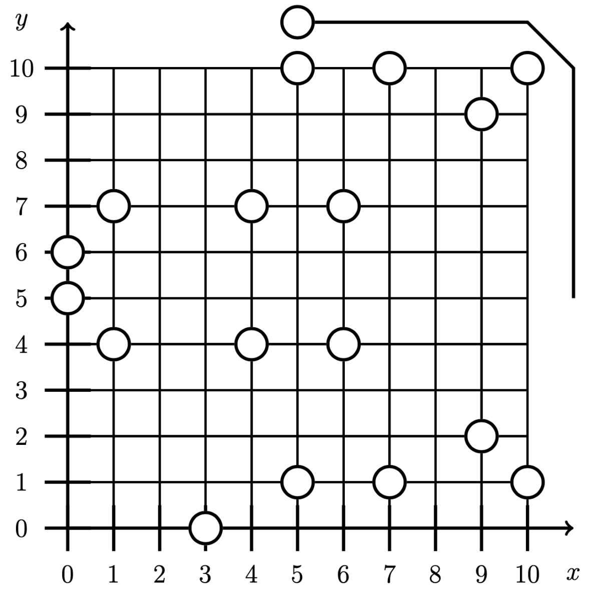


Figure 2: Elliptic curve $y^2 \equiv x^3 + x + 3$ mod 11

described in Section 3. The elements of the various sets on which the group acts are encoded as integers. For instance,

```
orbiter.out -linear_group -PGL 4 2 -end
```

creates the group $\mathrm{PGL}(4,2)$ acting on the 15 elements of $\mathfrak{G}r_1(\mathbb{F}_2^4)$. The basic types of groups are listed in Table 7.

For instance,

```
orbiter.out -v 3 -linear_group -PGGL 3 4 -end
```

creates $\mathrm{P\Gamma L}(3,4)$ acting on the 21 points of $\mathrm{PG}(2,4)$.

The command

```
orbiter.out -v 3 -linear_group -GL_d_q_wr_Sym_n 2 2 4 -end
```

creates the group $\mathrm{GL}(2,2) \wr \mathrm{Sym}(4)$ acting on $\mathrm{PG}(15,2)$ extended by a set of 20 extra points. The extra points are associated with the actions of the components of the wreath product: Four points form a permutation domain for the permutation part $\mathrm{Sym}(4)$. An additional $16 = 4 \times 4$ points form four permutation domains of $\mathrm{GL}(2,2)$, one for each factor.

A collineation of a projective space π is a bijective mapping from the points of π to themselves which preserves collinearity. That is, a collineation φ maps any three collinear points P, Q, R to another collinear triple $\varphi(P), \varphi(Q), \varphi(R)$. The collineations form a group with respect to composition, the collineation group. If M is the matrix of an endomorphism, then Ψ_M is the induced map on projective space. By considering the homomorphism $M \mapsto \Psi_M$, the group

Command	Arguments	Group
`-GL`	$n\ q$	$\mathrm{GL}(n,q)$
`-GGL`	$n\ q$	$\Gamma\mathrm{L}(n,q)$
`-SL`	$n\ q$	$\mathrm{SL}(n,q)$
`-SSL`	$n\ q$	$\Sigma\mathrm{L}(n,q)$
`-PGL`	$n\ q$	$\mathrm{PGL}(n,q)$
`-PGGL`	$n\ q$	$\mathrm{P\Gamma L}(n,q)$
`-PSL`	$n\ q$	$\mathrm{PSL}(n,q)$
`-PSSL`	$n\ q$	$\mathrm{P\Sigma L}(n,q)$
`-AGL`	$n\ q$	$\mathrm{AGL}(n,q)$
`-AGGL`	$n\ q$	$\mathrm{A\Gamma L}(n,q)$
`-ASL`	$n\ q$	$\mathrm{ASL}(n,q)$
`-ASSL`	$n\ q$	$\mathrm{A\Sigma L}(n,q)$
`-GL_d_q_wr_Sym_n`	$d\ q\ n$	$\mathrm{GL}(d,q) \wr \mathrm{Sym}(n)$

Table 7: Basic types of Orbiter matrix groups

$\mathrm{GL}(n+1,q)$ of invertible endomorphisms becomes a subgroup of the group of collineations of $\mathrm{PG}(n,q)$. This is the projectivity group $\mathrm{PGL}(n+1,q)$. It is isomorphic to $\mathrm{GL}(n+1,q)/\mathbb{F}_q^\times$. Another source of collineations is this: Let $\Phi \in \mathrm{Aut}(\mathbb{F}_q)$ be a field automorphism. Then Φ acts on projective space by sending $P(\mathbf{x})$ to $P(\mathbf{x}\Phi)$. This map is another type of collineation, called automorphic collineation. This way, $\mathrm{Aut}(\mathbb{F}_q)$ can be considered another subgroup of the group of collineations. If $q = p^h$ for some prime p and some integer h then

$$\Phi_0 : \mathbb{F}_q \to \mathbb{F}_q,\ x \mapsto x^p$$

is a generator for the cyclic group $C_h \simeq \mathrm{Aut}(\mathbb{F}_q)$. The collineation group of $\mathrm{PG}(n,q)$ ($n \geq 2$) is isomorphic to the semidirect product of the projectivity group and the automorphism group of the field. The collineation group is $\mathrm{P\Gamma L}(n+1,q) = \mathrm{PGL}(n+1,q) \ltimes \mathrm{Aut}(\mathbb{F}_q)$. We use the following notation for elements of $\mathrm{P\Gamma L}(n+1,q)$. Let Φ_0 be a generator for $\mathrm{Aut}(\mathbb{F}_q)$ and let $M \in \mathrm{GL}(n+1,q)$. The map

$$(\Psi_M, \Phi_0^k) : \mathrm{PG}(n,q) \to \mathrm{PG}(n,q),\ P(\mathbf{x}) \mapsto P(\mathbf{y}), \quad \mathbf{y} = (\mathbf{x} \cdot M)^{\Phi_0^k}$$

is denoted as

$$M_k. \tag{1}$$

The identity element is I_0, where I is the identity matrix and 0 is the residue class modulo h. The rules for multiplication and inversion in the collineation group are given as

$$M_k \cdot N_l = \Big(M \cdot N^{\Phi^{-k}}\Big)_{k+l}, \tag{2}$$

$$\Big(M_k\Big)^{-1} = \Big(\Big(M^{-1}\Big)^{\Phi^k}\Big)_{-k}. \tag{3}$$

The affine group $\mathrm{AGL}(n, q)$ is the semidirect product of $\mathrm{GL}(n, q)$ with $\mathbb{F}_q^n$. The affine semi-linear group $\mathrm{A\Gamma L}(n, q)$ is the semidirect product of $\mathrm{AGL}(n, q)$ with $\mathrm{Aut}(\mathbb{F}_q)$. The elements of $\mathrm{A\Gamma L}(n, q)$ are triples

$$(M, \mathbf{a}, k) \in \mathrm{GL}(n, q) \times \mathbb{F}_q^n \times \mathrm{Aut}(\mathbb{F}_q),$$

which act on $\mathbb{F}_q^n$ like so:

$$\Big(\mathbf{x}, (M, \mathbf{a}, k)\Big) \mapsto \big(\mathbf{x} \cdot M + \mathbf{a}\big)^{\Phi^k}.$$

We abbreviate the group elements as

$$M_{\mathbf{a},k} = (M, \mathbf{a}, k).$$

The multiplication in $\mathrm{A\Gamma L}(n, q)$ is

$$M_{\mathbf{a},k} \cdot N_{\mathbf{b},l} = (MN)_{\mathbf{a}N^{\Phi^{-k}} + \mathbf{b}^{\Phi^{-k}}, k+l}.$$

The inverse of an element is

$$\Big(M_{\mathbf{a},k}\Big)^{-1} = \Big(M^{-1}\Big)_{\mathbf{a}^{\Phi^k} M^{-1}, -k}.$$

A correlation is a one-to-one mapping between the set of points and the set of hyperplanes which reverses incidence. So, if ρ is a correlation and P is a point and ℓ is a hyperplane then P^ρ is a hyperplane and ℓ^ρ is a point and

$$\ell^\rho \in P^\rho \iff P \in \ell.$$

A correlation of order two is called polarity. The standard polarity is the map

$$\rho : \mathcal{P} \leftrightarrow \mathcal{L},\ P(\mathbf{x}) \leftrightarrow [\mathbf{x}].$$

It is possible to create various subgroups of a group. Table 8 lists some commands to do so. For instance, the command

```
orbiter.out -v 3 -linear_group -PGL 7 11 -Janko1 -end
```

creates the first Janko group as a subgroup of $\mathrm{PGL}(7, 11)$.

The following command

```
orbiter.out -v 3 -linear_group -PGL 3 11 -singer 1 -end
```

can be used to create a subgroup known as the Singer cycle. The Singer cycle in $\mathrm{GL}(d, q)$ is a generator for a subgroup of order $q^d - 1$. It induces an element of order $\frac{q^d-1}{q-1}$ on the associated projective geometry $\mathrm{PG}(d - 1, q)$. The additonal integer parameter k after the `-singer` command can be used to create a the subgroup of index k of the Singer cycle.

The command

Modifier	Arguments	Meaning
`-Janko1`		first Janko group, needs $\mathrm{PGL}(7, 11)$
`-monomial`		subgroup of monomial matrices
`-diagonal`		subgroup of diagonal matrices
`-null_polarity_group`		null polarity group
`-symplectic_group`		symplectic group
`-singer`	k	subgroup of index k in the Singer cycle
`-singer_and_frobenius`	k	subgroup of index k in the Singer cycle, extended by the Frobenius automorphism of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$
`-borel_upper`		Borel subgroup of upper triangular matrices
`-borel_lower`		Borel subgroup of lower triangular matrices
`-identity_group`		identity subgroup
`-subgroup_from_file`	f l	read subgroup from file f and give it the label l
`-orthogonal`	ϵ	orthogonal group O^ϵ, with $\epsilon \in \{\pm 1\}$ when n is even
`-subgroup_by_generators`	l o n s_1 ... s_n	Generate a subgroup from generators. The label "l" is used to denote the subgroup; o is the order of the subgroup; n is the number of generators and s_1, ..., s_n are the generators for the subgroup in string representation.

Table 8: Commands for creating subgroups

Modifier	Arguments	Meaning
`-wedge`		action on the exterior square
`-PGL2OnConic`		induced action of $\mathrm{PGL}(2, q)$ on the conic in the plane $\mathrm{PG}(2, q)$
`-subfield_structure_action`	s	action by field reduction to the subfield of index s
`-on_k_subspaces`	k	induced action on k dimensional subspaces
`-on_tensors`		induced action of $\mathrm{GL}(d, q) \wr \mathrm{Sym}(n)$ on the tensor space
`-on_rank_one_tensors`		induced action of $\mathrm{GL}(d, q) \wr \mathrm{Sym}(n)$ on the tensor space

Table 9: Commands for creating induced or restricted group actions

```
orbiter.out -v 3 -linear_group -PGL 3 11 -singer_and_frobenius 19 -end
```

can be used to create a subgroup of index 19 of the Singer cycle of PG$(2, 11)$, extended by a group of order 3 that arises from the field extension $\mathbb{F}_{11}^3$ over $\mathbb{F}_{11}$. The group created by this command has order 21.

It is possible to create new group actions from old. Table 9 lists some commmands to do so. For instance, the command

```
orbiter.out -v 4 \
  -linear_group -GL_d_q_wr_Sym_n 2 2 4 -on_tensors -end
```

create the group GL$(2, 2) \wr$ Sym(4) acting on the 65535 elements of PG$(15, 2)$. The auxilliary point that result from the wreath product action are blended out. This is done by restricting the action to the points of PG$(15, 2)$.

It is possible to perform group theoretic tasks using the `-group_theoretic_activities` option. The possible tasks are determined through the options described in Table 10.

The command

```
orbiter.out -v 4 \
  -linear_group -GL_d_q_wr_Sym_n 2 2 4 -on_tensors -end \
  -group_theoretic_activities \
  -orbits_on_points
pdflatex GL_2_2_wreath_Sym4_res65535_orbits.tex
open GL_2_2_wreath_Sym4_res65535_orbits.pdf
```

create the group GL$(2, 2) \wr$ Sym(4) acting on the 65535 elements of PG$(15, 2)$. The command also computes the orbit on points. The latex file `GL_2_2_wreath_Sym4_res65535_orbits.tex` is created. It contains a list of the generators of the group and the orbits.

Through its interface to Magma [5], Orbiter can be used to compute the conjugacy classes of groups. For instance, the command

```
orbiter.out -v 3 -linear_group -PSL 3 2 -end \
  -group_theoretic_activities -classes
```

can be used to create a report about the conjugacy classes of the simple group PSL$(3, 2)$.

The quaternion group is a group of order 8 generated by the following matrices over $\mathbb{R}$:

$$i = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad j = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}, \quad k = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

It is isomorphic to a subgroup of SL$(2, 3)$. To perform the embedding, we need to replace the real number -1 by the corresponding field element in $\mathbb{F}_3$. Recalling the convention from Section 3 that Orbiter field elements are integers in the interval $[0, \ldots, q-1]$, we replace -1 by 2. The Orbiter command

Modifier	Arguments	Meaning
`-orbits_on_subsets`	k	Compute orbits on k-subsets
`-orbits_on_points`		Compute orbits in the action that was created
`-orbits_of`	i	Compute orbit of point i in the action that was created
`-stabilizer`		Compute the stabilizer of the orbit representative (needs -orbits_on_points)
`-draw_poset`		Draw the poset of orbits (needs -orbits_on_subsets)
`-classes`		Compute a report of the conjugacy classes of elements (needs Magma [5])
`-normalizer`		Compute the normalizer (needs Magma [5]; needs a group with a subgroup)
`-report`		Produce a latex report about the group
`-sylow`		Include Sylow subgroups in the report (needs -report)
`-print_elements`		Produce a printout of all group elements
`-print_elements_tex`		Produce a latex report of all group elements
`-group_table`		Produce the group table (needs -report)
`-orbits_on_set_system_from_file`	fname f l	reads the csv file “fname” and extract sets from columns $[f, ..., f + l - 1]$
`-orbit_of_set_from_file`	fname	reads a set from the text file “fname” and computes orbits on the elements of the set
`-multiply`	s_1 s_2	Creates group elements from s_1 and s_2 and multiplies
`-inverse`	s	Creates a group element from s and computes its inverse
`-tensor_classify`	d	Classify tensors up to rank d
`-tensor_permutations`		Compute permutation representation of generators of wreath product

Table 10: Group theoretic activities

```
orbiter.out -v 3 -linear_group -SL 2 3 \
    -subgroup_by_generators "quaternion" "8" 3 \
    "1,1,1,2" \
    "2,1,1,1" \
    "0,2,1,0" \
    -end \
    -group_theoretic_activities  \
    -print_elements_tex \
    -group_table \
    -report \
-end
```

creates the group. The command produces a list of group elements.

The group of the cube can be created over the field $\mathbb{F}_3$ like so:

```
orbiter.out -v 3 -linear_group -GL 3 3 \
  -subgroup_by_generators "cube" "48" 3 \
    "0,1,0,2,0,0,0,0,1" \
    "0,0,1,0,1,0,2,0,0" \
    "2,0,0,0,1,0,0,0,1" \
    -end \
    -group_theoretic_activities \
    -print_elements_tex \
    -report
```

The tetrahedral subgroup can be created like so:

```
orbiter.out -v 3 -linear_group -GL 3 3 \
    -subgroup_by_generators "tetra" "12" 2 \
    "0,1,0,0,0,1,1,0,0" \
    "0,0,1,2,0,0,0,2,0" \
    -end \
    -group_theoretic_activities \
    -print_elements_tex \
    -report
```

Sometimes, the generators depend on specific choices made for the finite field. For instance, if the field if a true extension field over its prime field, the choice of the polynomial matters. This is particularly relevant if generators are taken from other sources. For instance, the electronic Atlas of finite simple groups [15] lists generators for $U_3(3)$ as 3×3 matrices over the field $\mathbb{F}_9$ using the following short Magma [5] program:

```
F<w>:=GF(9);
x:=CambridgeMatrix(1,F,3,[
"164",
"506",
"851"]);
y:=CambridgeMatrix(1,F,3,[
"621",
"784",
"066"]);
G<x,y>:=MatrixGroup<3,F|x,y>;
```

The generators are given using the Magma command `CambridgeMatrix`, which allows for more efficient coding of field elements. The field elements are coded as base-3 integers (like in Orbiter) with respect to the Magma version of $\mathbb{F}_9$. Magma uses Conway polynomials to generate finite fields which are not of prime order. The Conway polynomial for $\mathbb{F}_9$ can be determined using the following Magma command (which can be typed into the Magma online calculator at [14])

```
F<w>:=GF(9);
print DefiningPolynomial(F);
```

which results in

```
$.1^2 + 2*$.1 + 2
```

which is the Magma way of printing the polynomial $X^2 + 2X + 2$. To have Orbiter use this polynomial, the `-override_polynomial` option can be used. First, the polynomial is identified with the vector of coefficients $(1, 2, 2)$ which is then read as base-3 representation of an integer as

$$(1, 2, 2) = 1 \cdot 3^2 + 2 \cdot 3 + 2 = 17.$$

The Orbiter command

```
orbiter.out -v 3 -linear_group -override_polynomial "17" -PGL 3 9 \
    -subgroup_by_generators "U_3_3" "6048" 2 \
    "1,6,4, 5,0,6, 8,5,1" \
    "6,2,1, 7,8,4, 0,6,6" \
    -end  \
-group_theoretic_activities \
    -report -end
```

can then be used to create the group. Notice how the generators are encoded almost like in the Magma command, except that commas are used to separate entries.

7 Orbits on Subsets

The lattice of subsets of a set X is $\mathfrak{P}(X)$, the set of all subsets of X, ordered with respect to inclusion. For instance, Figure 3 shows the lattice of subsets of a 4-element set. Assume that a group G acts on X, and hence on the lattice by means of the induced action on subsets.

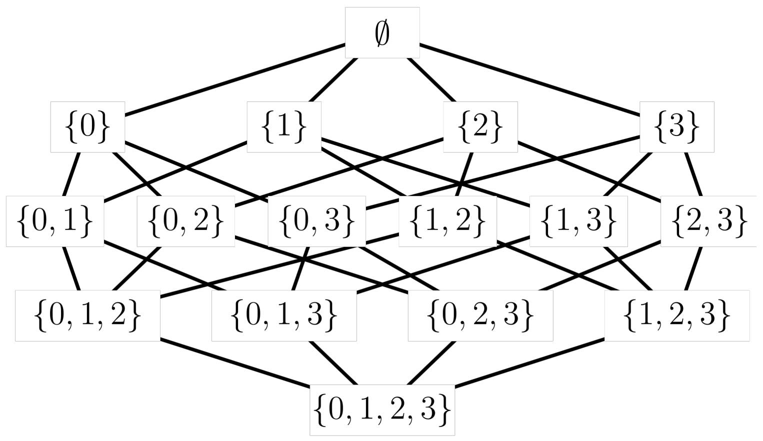


Figure 3: The lattice of subsets of a 4-element set

The orbits of G on subsets clump together nodes in the lattice. The set of G-orbits form a new poset, the poset of orbits. Poset classification is the process of computing the poset of orbits. Orbiter has an algorithm to perform poset classification. In many cases, we are not interested in the full lattice of subsets $\mathfrak{P}(X)$ but rather in a subposet of it. We require that the subposet is closed under the group action and that the following property holds:

$$x, y \in \mathfrak{P}(X) \text{ and } x \leq y \ \Rightarrow \ \Big(y \in \mathcal{P} \to x \in \mathcal{P} \Big).$$

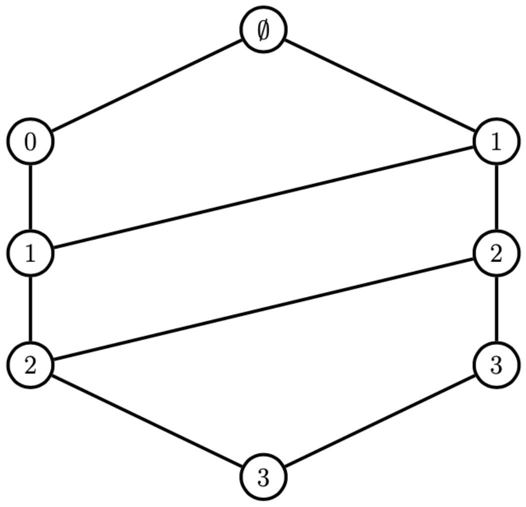
The join of two subsets in the poset may or may not belong to the poset. Let us consider the poset of subsets of the 4-element set under the action of a group of order 3. We take the 4 points to be the vectors of $X = \mathbb{F}_2^2$. Let G be the group generated by the Singer cycle in $\mathrm{GL}(2,2)$, so

$$G = \langle \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \rangle \simeq \langle (0)(1,3,2) \rangle,$$

the latter being the permutation representation on the set X. Thus, G is a group of order 3 acting with one fixed point. The command

```
orbiter.out -v 3 -linear_group -GL 2 2 -singer 1 -end  \
    -group_theoretic_activities \
    -orbits_on_subsets 4 \
    -draw_poset \
    -report
```

computes the orbits of G on the poset of subsets. The poset of orbits is shown in Figure 4. All nodes except for the root node are labeled by elements of $X = \{0, 1, 2, 3\}$. In order to determine the set that is associated to a node, we follow the unique leftmost path to the root node and collect the node lables. This produces the set associated to the node. The orbit representatives are indicated next to the diagram. By convention, the order of the stabilizer is written as a subscript. Since this example is small enough, it is possible to show



$\{\}_3$
$\{0\}_3$
$\{1\}_1$
$\{0,1\}_1$
$\{1,2\}_1$
$\{0,1,2\}_1$
$\{1,2,3\}_3$
$\{0,1,2,3\}_3$

Figure 4: The poset of orbits under the Singer group

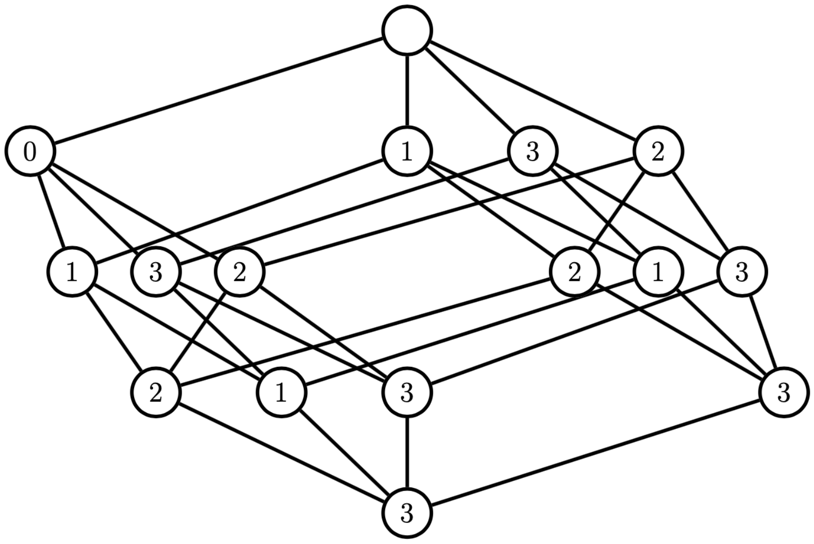


Figure 5: The poset with orbits indicated by grouping

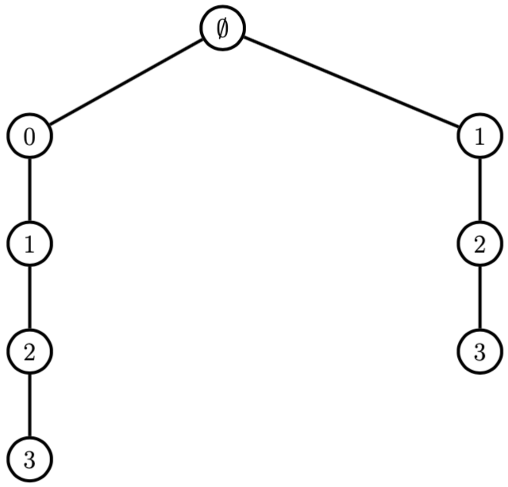


Figure 6: The lex-least spanning tree for the poset of orbits

the complete orbits, as in Figure 5. Elements belonging to an orbit are grouped together. The leftmost element in each group is the orbit representative.

As the poset of orbits can get quite busy, it is often desired to replace it by the lex-least spanning tree. This is the tree that results by keeping only the links between a node and its lex-least ancestor, see Figure 6.

8 Orbits on Subspaces

Orbiter can compute the orbits of a group on the lattice of subspaces of a finite vector space.

Suppose we want to classify the subspaces in PG$(3,2)$ under the action of the orthogonal group. The orthogonal group is the stabilizer of a quadric. In PG$(3,2)$ there are two distinct nondegenerate quadrics, $\mathcal{Q}^+(3,2)$ and $\mathcal{Q}^-(3,2)$. The $\mathcal{Q}^+(3,2)$ quadric is a finite version of the quadric given by the equation

$$x_0x_1 + x_2x_3 = 0,$$

and depicted over the real numbers in Figure 7. PG$(3,2)$ has 15 points:

$P_0 = (1,0,0,0)$	$P_4 = (1,1,1,1)$	$P_8 = (1,1,1,0)$	$P_{12} = (0,0,1,1)$
$P_1 = (0,1,0,0)$	$P_5 = (1,1,0,0)$	$P_9 = (1,0,0,1)$	$P_{13} = (1,0,1,1)$
$P_2 = (0,0,1,0)$	$P_6 = (1,0,1,0)$	$P_{10} = (0,1,0,1)$	$P_{14} = (0,1,1,1)$
$P_3 = (0,0,0,1)$	$P_7 = (0,1,1,0)$	$P_{11} = (1,1,0,1)$	

The $\mathcal{Q}^+(3,2)$ quadric given by the equation above consists of the nine points

$$P_0, P_1, P_2, P_3, P_4, P_6, P_7, P_9, P_{10}.$$

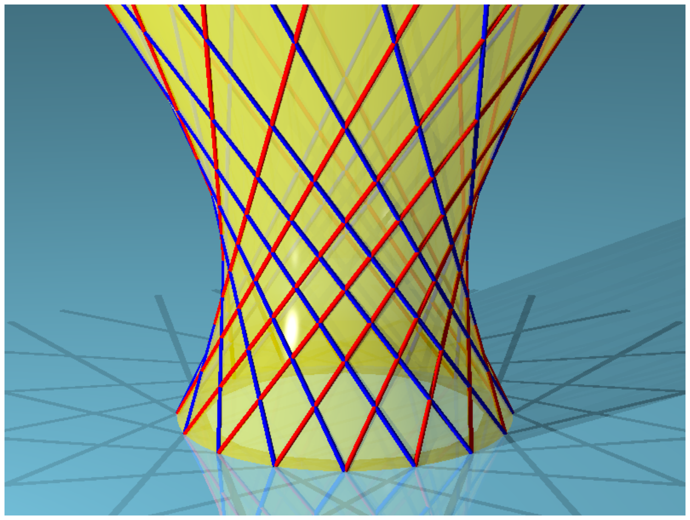


Figure 7: The hyperbolic quadric in affine space $\mathbb{R}^3$

The quadric is stabilized by the group $\mathrm{PGO}^+(4,2)$ of order 72. The command

```
orbiter.out -v 5 -linear_group -PGL 4 2 -orthogonal 1 -end \
  -group_theoretic_activities -orbits_on_subspaces 4 \
  -draw_poset
```

produces a classification of all subspaces of $\mathrm{PG}(3,2)$ under $\mathrm{PGO}^+(4,2)$. The option `-draw_poset` creates a Hasse diagram of the classification as shown Figure 8. The Hasse diagram is the poset of orbits of the group on the subspace lattice.

9 Graph Theory

Orbiter is able to work with algebraically defined graphs. It can also construct and classify graphs up to isomorphism. In Table 11, command line arguments are shown for some of the graphs that Orbiter can create. For instance,

```
orbiter.out -v 2 -create_graph -Johnson 5 2 0 -end -save graph_J520.bin
```

creates $J(5,2,0)$, also known as the Petersen graph.

```
orbiter.out -v 2 -create_graph  -Paley 13 -end -save graph_P13.bin
```

creates the Paley graph of order 13. Regarding the problem of classifying small graphs, Table 12 lists the command line options that are available. Here is an example:

```
orbiter.out -v 2 -graph_classify -n 4
```

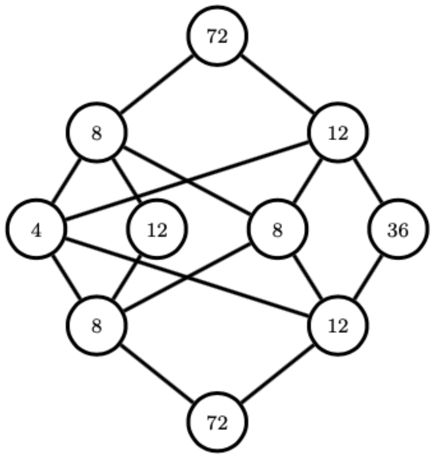


Figure 8: Hasse-diagram for the orbits of the orthogonal group $O^+(4,2)$ on subspaces of PG$(3,2)$

Key	Arguments	Meaning
`-Johnson`	$n\ k\ s$	Johnson graph
`-Paley`	q	Paley graph
`-Sarnak`	$p\ q$	Lubotzky-Phillips-Sarnak graph [10]
`-Schlaefli`	q	Schlaefli graph
`-Shrikhande`		Shrikhande graph
`-Winnie_Li`	$q\ i$	Winnie-Li graph [9]
`-Grassmann`	$n\ k\ q\ r$	Grassmann graph
`-coll_orthogonal`	$\epsilon\ d\ q$	Collinearity graph of $O^\epsilon(d,q)$

Table 11: Types of graphs

Option	Arguments	Meaning
`-girth`	d	Girth at least d
`-regular`	r	Regular of degree r
`-no_transmitter`		Tournament without transmitter (requires `-tournament`)

Table 12: Options for classifying graphs

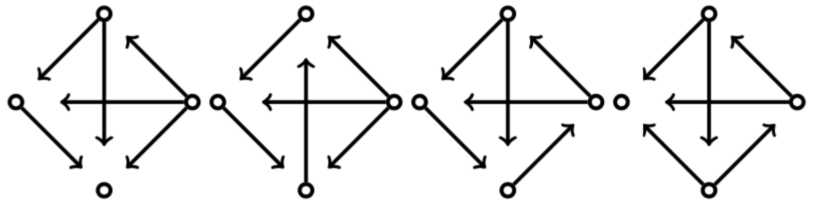


Figure 9: The four isomorphism types of tournaments on 4 vertices

classifies all graphs with 4 vertices. For this, the set $X = \binom{V}{2}$ is considered, where $V = \{0, 1, \ldots, n-1\}$. The poset $\mathcal{P}$ is the lattice of subsets of X. The group action of $\mathrm{Sym}(V)$ induces an action on the lattice. For tournaments, the option `-tournament` can be added. In this case, $X = V^{[2]}$ is the set of ordered pairs of elements from V. The ordered pair (a, b) represents the fact that a beats b in the tournament. Again, the poset $\mathcal{P}$ is the lattice of subsets of X and the group $\mathrm{Sym}(V)$ acts in the induced action on ordered pairs. For example,

```
orbiter.out -v 2 -graph_classify -n 4 -v 2 -tournament -draw_graphs_at_level 6
```

classifies the tournaments on 6 vertices. The `-draw_graphs_at_level 6` option instructs Orbiter to draw all representatives at level 6. Figure 9 shows the resulting list of 4 tournaments.

10 Projective Geometry

Orbiter can be used to classify sets in projective space and to compute their collineation stabilizer. Here is an example. We consider the following three Orbiter commands:

```
orbiter.out -v 5 -create_combinatorial_object -q 11 \
  -elliptic_curve 1 3 -end \
  -save "./"
orbiter.out -v 2 -process_combinatorial_objects \
  -draw_points_in_plane EC_11_1_3 -q 11 -n 2 \
  -fname_base_out EC_11_1_3 -embedded \
  -input -file_of_points elliptic_curve_b1_c3_q11.txt \
  -end \
  -end
orbiter.out -v 2 -canonical_form_PG  2 11 \
  -input \
  -file_of_points elliptic_curve_b1_c3_q11.txt \
  -end \
  -prefix PG_2_11_EC \
  -save elliptic_curve_b1_c3_q11_classified \
  -report
```

The first command creates an elliptic curve over a finite field. Specifically, it creates the curve whose affine equation is

$$y^2 \equiv x^3 + x + 3 \bmod 11.$$

The curve turns out to have exactly 18 points in $\mathrm{PG}(2,11)$. The second command produces a picture of the point set in $\mathrm{PG}(2,11)$, shown in Figure 2. The third command computes the collineation stabilizer. This is done by using the techniques of canonical forms in graphs, using the Nauty [11] package which is included in Orbiter. Orbiter shows that the curve has a collineation stabilizer of order 6, generated by

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 8 \\ 5 & 9 & 5 \\ 8 & 1 & 1 \end{bmatrix}.$$

The types of objects that can be created using `-create_combinatorial_object` are listed in Tables 13 and 14. Modifier options that apply to multiple options are listed in Table 15.

The purpose of the `-process_combinatorial_objects` option is to perform various jobs. The input and output are data streams. A data stream can represents integer data. Integer data can encode sets in projective space, for instance. The types of jobs are listed in Table 16.

11 Cubic Surfaces

Orbiter can classify cubic surfaces with 27 lines over finite fields. There are several different approaches to classify cubic surfces over finite fields with 27 lines under the collineation group $\mathrm{P\Gamma L}(4,q)$. One approach is described in [4] and relies on Schlaefli's notion of a double six as a substructure [13]. Another approach is through non-conical six-arcs in a plane, as described in [8]. Both approaches have been implemented in Orbiter. The purpose of the construction algorithm is to produce the equations of surfaces. In order to do so, the notion of a double six of lines in $\mathrm{PG}(3,q)$ is used. A double six determines a unique surfaces but a surface may have several double sixes associated to it. The classification algorithms sorts out the relationship between the isomorphism types of double sixes and the isomorphism types of cubic surfaces. In order to classify all double sixes, yet another substructure is considered. These are the five-plus-ones. They consist of 5 lines with a common transversal. The poset classification algorithm is used to classify the five-plus-ones. Also, Orbiter will sort out the isomorphism classes of double sixes based on their relation to the five-plus-ones. In order to classify the five-plus-ones, the related Klein quadric is considered. Lines in $\mathrm{PG}(3,q)$ correspond to points on the Klein quadric. Thus, the five-plus-one configurations of lines correspond to certain configurations of points on the Klein quadric.

The command

Key	Arguments	Purpose
`-hyperoval`		To create a hyperoval
`-subiaco_oval`	f_short	Subiaco oval
`-subiaco_hyperoval`		Subiaco hyperoval
`-adelaide_hyperoval`		Adalaide hyperoval
`-translation`	exponent	translation hyperoval
`-Segre`		Segre hyperoval
`-Payne`		Payne hyperoval
`-Cherowitzo`		Cherowitzo hyperoval
`-OKeefe_Penttila`		OKeefe, Penttila hyperoval
`-BLT_database`	k	The kth BLT-set of order q from the database $(k = 0, 1, \ldots)$
`-ovoid`		ovoid
`-Baer`		Baer subgeometry
`-orthogonal`	ϵ	quadric of ϵ-type
`-hermitian`		hermitian variety given by $\sum_{i=0}^{n} X_i^{\sqrt{q}+1} = 0$
`-cubic`		cubic
`-twisted_cubic`		twisted cubic
`-elliptic_curve`	a b	elliptic curve $y^2 = x^3 + ax + b$
`-ttp_constru ction_A`		twisted tensor product code of type A
`-ttp_constru ction_A_hyperoval`		twisted tensor product code of type A
`-ttp_constru ction_B`		twisted tensor product code of type B

Table 13: Orbiter Objects (Part 1)

Key	Arguments	Purpose
`-unital_XXq_YZq_ZYq`		unital with equation $XX^q + YZ^q + ZY^q = 0$
`-desarguesian_line_spread_in_PG_3_q`		desarguesian line spread in $\mathrm{PG}(3, q)$
`-Buekenhout_Metz`		Buekenhout Metz unital
`-Uab`	a b	Buekenhout Metz unital in the form of Barwick and Ebert [1]
`-whole_space`		whole space
`-hyperplane`	pt	hyperplane given by dual coordinates associated with the given point
`-segre_variety`	a b	Segre variety
`-Maruta_Hamada_arc`		Maruta Hamada arc
`-projective_variety`	l d $\mathcal{C}$	Projective variety of degree d with label l, with coefficient vector $\mathcal{C}$
`-projective_curve`	l r d $\mathcal{C}$	Projective curve of degree d with label l, with coefficient vector $\mathcal{C}$ in r variables

Table 14: Orbiter Objects (Part 2)

Key	Arguments	Purpose
`-q`	q	The size of the finite field $\mathbb{F}_q$
`-Q`	Q	The field size of the extension field $\mathbb{F}_Q$
`-n`	n	The projective dimension
`-poly`	r	Use polynomial with rank r to create the field $\mathbb{F}_q$
`-poly_Q`	r	Use polynomial with rank r to create the field $\mathbb{F}_Q$
`-embedded_in_PG_4_q`		
`-BLT_in_PG`		BLT set with point ranks in PG

Table 15: Orbiter Objects: Modifiers

Job key	Purpose
`-dualize_hyperplanes_to_points`	Turns ranks of hyperplanes into ranks of points
`-dualize_points_to_hyperplanes`	Turns ranks of points into ranks of hyperplanes
`-ideal`	Compute the ideal of a set of points
`-homogeneous_polynomials`	Prints the equation whose coefficient vector is the input vector
`-canonical_form`	Computes the canonical form of a set
`-draw_points_in_plane`	Produces a drawing of a set of points in a projective plane
`-klein`	Applies the Klein correspondence
`-line_type`	Computes the line type
`-plane_type`	Computes the plane type
`-conic_type`	Computes the conic type
`-hyperplane_type`	Computes the hyperplane type
`-intersect_with_set_from_file`	Computes the intersection with a set specified in a file
`-arc_with_given_set_as_s_lines_after_dualizing`	Finds arcs with the given set as s-lines
`-arc_with_two_given_sets_of_lines_after_dualizing`	Finds arcs with the two given sets as s-lines and t-lines, respectively
`-arc_with_three_given_sets_of_lines_after_dualizing`	Finds arcs with the three given sets as s-lines and t-lines and u-lines, respectively

Table 16: Orbiter Jobs in PG(n, q)

```
orbiter.out -v 3 -linear_group -PGL 4 7 -wedge -end \
  -group_theoretic_activities -surface_classify -end
```

classifies all cubic surfaces over the field $\mathbb{F}_7$ under the projective linear group. If desired, it is possible to use

```
orbiter.out -v 3 -linear_group -PGGL 4 4 -wedge -end \
  -group_theoretic_activities -surface_classify -end
```

to perform the same classification with respect to the collineation group $\mathrm{P\Gamma L}(4,4)$. The `-report` option can be used to create a report of the classified surfaces. So, for instance

```
orbiter.out -v 3 -linear_group -PGGL 4 4 -wedge -end \
  -group_theoretic_activities -surface_classify -report
```

produces a latex report of the surface in $\mathrm{PG}(3,4)$. The `-surface_recognize` option can be used to identify a given surface in the list produced by the classification. For instance,

```
orbiter.out -v 3 -linear_group -PGGL 4 8 -wedge -end \
    -group_theoretic_activities -surface_recognize -q 8 \
    -by_coefficients "1,6,1,8,1,11,1,13,1,19" -end -end
```

identifies the surface (cf. Table 5)

$$X_0^2X_3 + X_1^2X_2 + X_1X_2^2 + X_0X_3^2 + X_1X_2X_3 = 0 \tag{4}$$

in the classification of surfaces over the field $\mathbb{F}_8$. This means that an isomorphism from the given surface to the surface in the list is computed. Also, the generators of the automorphism group of the given surface are computed, using the known generators for the automorphism group of the surface in the classification. For instance, executing the command above creates an isomorphism between the given surface and the surface in the catalogue:

$$\begin{bmatrix} 1 & 4 & 4 & 0 \\ 6 & 0 & 0 & 0 \\ 6 & 2 & 0 & 1 \\ 7 & 0 & 4 & 0 \end{bmatrix}_0 . \tag{5}$$

Orbiter can compute isomorphism between two given surfaces. The surfaces must have 27 lines. For instance, the command

```
orbiter.out -v 3 -linear_group -PGGL 4 8 -wedge -end \
    -group_theoretic_activities -surface_isomorphism_testing \
      -q 8 -by_coefficients \
        "5,5,5,8,5,9,5,10,5,11,5,12,4,14,4,15,1,18,1,19" -end \
      -q 8 -by_coefficients "1,6,1,8,1,11,1,13,1,19" -end
```

computes an isomorphism between the two $\mathbb{F}_8$-surfaces

$$\begin{aligned}0 &= \alpha^3 X_0^2 X_2 + \alpha^3 X_1^2 X_2 + \alpha^3 X_1^2 X_3 + \alpha^3 X_0 X_2^2 + \alpha^3 X_1 X_2^2 + \alpha^3 X_2^2 X_3\\ &\quad + \alpha^2 X_1 X_3^2 + \alpha^2 X_2 X_3^2 + X_0 X_2 X_3 + X_1 X_2 X_3,\\ 0 &= X_0^2 X_3 + X_1^2 X_2 + X_1 X_2^2 + X_0 X_3^2 + X_1 X_2 X_3.\end{aligned}$$

The isomorphism is given as a collineation:

$$\left[\begin{array}{cccc} 2 & 3 & 0 & 0\\ 7 & 3 & 0 & 0\\ 0 & 0 & 3 & 0\\ 2 & 3 & 2 & 4 \end{array}\right]_2 .$$

In here, the numerical representation of elements of $\mathbb{F}_8$ as integers in the interval $[0, 7]$ is used. The exponent of the Frobenius automorphism is listed as a subscript.

A second algorithm to classify cubic surfaces has been described in [3] and in [8]. This algorithm is available in Orbiter also. For instance, the command

```
orbiter.out -v 3 -linear_group -PGL 4 13 -end \
    -group_theoretic_activities \
    -classify_surfaces_through_arcs_and_trihedral_pairs 13
```

classifies all cubic surfaces with 27 lines over the field $\mathbb{F}_{13}$ using this algorithm. A report of the classification is produced in the file `arc_lifting_q13.tex`.

Besides classification, there are two further ways to create surfaces in Orbiter. The first is a built-in catalogue of cubic surfaces with 27 lines for small finite fields $\mathbb{F}_q$ (at the moment, $q \leq 101$ is required). The second is a way of creating members of known infinite families. Both are facilitated using the `-create_surface` option. For instance,

```
orbiter.out -v 3 -linear_group -PGL 4 13 -wedge -end \
    -group_theoretic_activities \
    -create_surface -family_S 3 -q 13 -end
```

creates the member of the Hilbert, Cohn-Vossen surface described in [4] with parameter $a = 3$ and $b = 1$ over the field $\mathbb{F}_{13}$. The command

```
orbiter.out -v 3 -linear_group -PGGL 4 8 -wedge -end \
    -group_theoretic_activities \
    -create_surface -q 4 -catalogue 0 -end
```

creates the unique cubic surface with 27 lines over the field $\mathbb{F}_4$ which is stored under the index 0 in the catalogue. It is possible to apply a transformation to the surface. Suppose we are interested in the surface over $\mathbb{F}_8$ created in (4). The command

```
orbiter.out -v 3 -linear_group -PGGL 4 8 -wedge -end \
    -group_theoretic_activities \
    -create_surface -q 8 -catalogue 0 -end \
    -transform_inverse "1,4,4,0,6,0,0,0,6,2,0,1,7,0,4,0,0"
```

creates surface 0 over $\mathbb{F}_8$ and applies the inverse transformation to recover the surface whose equation was given in (4). The surface number 0 over $\mathbb{F}_8$ is created, and the transformation (5) is applied in inverse. The commands `-transform` and `-transform_inverse` accept the transformation matrix in row-major ordering, with the field automorphism as additional element. It is possible to give a sequence of transformations. In this case, the transformations are applied in the order in which the commands are given on the command line.

12 Arcs in Projective Planes

A (k, d)-arc in a projective plane π is a set S of k points such that very line intersects S in at most d points. Arcs are related to linear codes and other structures. Two arcs S_1 and S_2 are equivalent if there is a projectivity Φ such that $\Phi(A) = B$. The problem of classifying arcs is the problem of determining the orbits of the projectivity group on arcs. At times, we consider the larger group of collineations. In that case, the problem of classifying arcs is the problem of determining the orbits of the collineation group on arcs. Orbiter can solve such classification problems, at least for small parameter cases. Here is an example. A hyperoval in a plane $\mathrm{PG}(2, 2^e)$ is a $(2^e + 2, 2)$-arc. It is interesting to classify the hyperovals up to collineation equivalence under the group $\mathrm{P\Gamma L}(3, 2^e)$. The command

```
orbiter.out -v 4 \
    -linear_group -PGGL 3 16 -end \
    -group_theoretic_activities \
      -classify_arcs 18 \
      -classify_arcs_d 2  \
      -exact_cover \
        -input_prefix ./ARCS/ \
        -output_prefix ./SYSTEMS/ \
        -solution_prefix ./SOLUTIONS/ \
        -starter_size 18 \
        -base_fname arcs_q16_d2 \
        -lex \
      -end \
    -end
```

performs the classification of hyperovals in $\mathrm{PG}(2, 16)$. There are exactly two hyperovals in this plane. Orbiter also finds the stabilizers of these arcs. They have orders 16320 and 144, respectively.

13 The Povray Interface

Orbiter can be used to create high quality raytracing graphics. Orbiter serves as a front end for 3D graphics processed trough Povray [12]. This is a multi step process: A 3D scene is defined through orbiter commands. Next, Orbiter produces Povray files. After that, the povray files are processed through povray, and turned into graphics files (png), called frames. The frames can be turned into a video by using tools like ffmpeg. Tables 17 and 18 summarizes the Orbiter commands to build objects of a 3D scene. Building the scene itself does not create any graphical output. To this end, the commands in Table 19 are used. Each of these commands applies to a group of objects of the same kind. Groups of objects are created using the commands in Table 18 which start with `group_of`. Here is a simple example which combines scene building and graphical output. The example creates a cube with vertices, edges and faces:

```
orbiter.out -v 2 -povray \
    -round 0 -nb_frames_default 30 -output_mask cube_%d_%03d.pov \

    -video_options -W 1024 -H 768  -global_picture_scale 0.5 \
      -default_angle 75 -clipping_radius 2.7 \
    -end \
    -scene_objects \
      -obj_file cube_centered.obj \
      -edge "0, 1" \
      -edge "0, 2" \
      -edge "0, 4" \
      -edge "1, 3" \
      -edge "1, 5" \
      -edge "2, 3" \
      -edge "2, 6" \
      -edge "3, 7" \
      -edge "4, 5" \
      -edge "4, 6" \
      -edge "5, 7" \
      -edge "6, 7" \
      -group_of_things_as_interval 0 8 \
      -spheres 0 0.3 "texture{ Polished_Chrome pigment{quick_color
 White} }" \
      -group_of_things_as_interval 0 6 \
      -prisms 1 0.05 "texture{ pigment{ color Yellow transmit 0.7
} finish {diffuse 0.9 phong 0.6} }" \
      -group_of_things_as_interval 0 12 \
      -cylinders 2 0.15 "texture{ pigment{ color Red  } finish {di
ffuse 0.9 phong 0.6} }" \
    -scene_objects_end \
    -povray_end

```

This command will tell Orbiter to create 30 povray files (extension .pov), one for each frame of a rotating scene. The scene containes a cube whose vertices are shown in chrome, whose

Command	Arguments	Purpose
`-cubic_lex`	coeffs	cubic surface given by 20 coefficients in lexicographic ordering
`-cubic_orbiter`	coeffs	cubic surface given by 20 coefficients in Orbiter ordering
`-cubic_Goursat`	A B C	cubic surface with tetrahedral symmetry given by 3 Goursat coefficients as $Axyz + B(x^2+y^2+z^2)+C=0$
`-quadric_lex_10`	coeffs	quadric surface given by 10 coefficients in lexicographic ordering
`-quartic_lex_35`	coeffs	quartic surface given by 35 coefficients in lexicographic ordering
`-point`	coeffs	point given by three coordinates
`-point_list_from_csv_file`	fname	List of points with coordinates given in a csv file
`-line_through_two_points_recentered_from_csv_file`	fname	List of lines through two points with point coordinates given in a csv file
`-line_through_two_points_from_csv_file`	fname	List of lines through two points with point coordinates given in a csv file
`-point_as_intersection_of_two_lines`	i_1 i_2	Create a point from the intersection of two lines i_1 and i_2
`-edge`	i_1 i_2	Create an edge (line segment) between points i_1 and i_2
`-text`	i_1 s	Create a label s located at the point i
`-triangular_face_given_by_three_lines`	i_1 i_2 i_3	Create a triangular face give by three lines i_1, i_2, i_3
`-face`	pts	Create a face through the vertices pts, ordered cyclically
`-quadric_through_three_skew_lines`	i_1 i_2 i_3	Create a quadric through three skew lines
`-plane_defined_by_three_points`	i_1 i_2 i_3	Create a plane through three noncollinear points
`-line_through_two_points_recentered`	pt-coords	Create a line through two points given by 6 coordinates, recentered
`-line_through_two_points`	pt-coords	Create a line through two points given by 6 coordinates

Table 17: Scene definition commands (part 1)

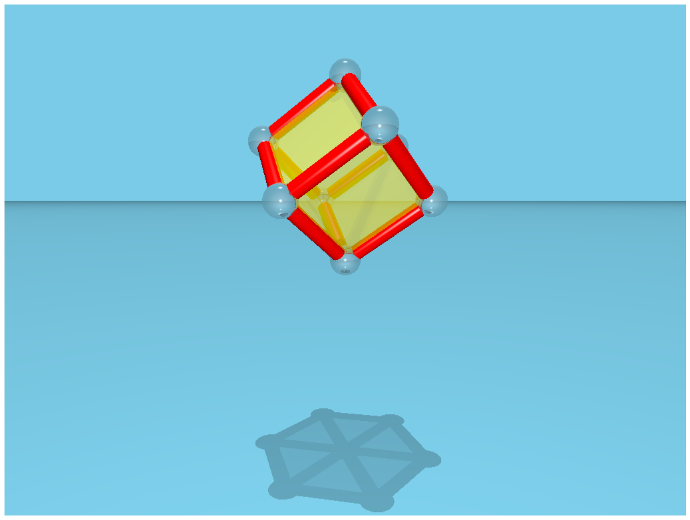
Command	Arguments	Purpose
`-line_through_two_existing_points`	i_1 i_2	Create a line through two points
`-line_through_point_with_direction`	coeffs6	Create a line through a point (x, y, z) with a given direction (u_x, u_y, u_z), where coeffs6 $= x, y, z, u_x, u_y, u_z$
`-plane_by_dual_coordinates`	coeffs4	Create a plane $ax + by + cz + d = 0$ give four dual coordinates as coeff4 $= a, b, c, d$
`-dodecahedron`		Create a Dodecahedron centered at the origin (20 points, 30 edges, 12 faces)
`-Hilbert_Cohn_Vossen_surface`		Create the Hilbert, Cohn-Vossen surface (1 cubic surface, 45 tritangent planes, 27 lines)
`-obj_file`	fname	Read points and faces from the given .obj file
`-group_of_ things`	list	Create a group of things from the given list
`-group_of_things_with_offset`	list offset	Create a group of things from the given list, each value is increase by offset
`-group_of_things_as_interval`	a b	Create a group of things from the interval $a, \ldots, a + b - 1$
`-group_of_things_as_interval_with_exceptions`	a b ex	Create a group of things from the interval $a, \ldots, a+b-1$ with the exceptional elements in the list ex removed
`-group_of_ all_points`		Create a group of things from all points currently defined
`-group_of_ all_faces`		Create a group of things from all faces currently defined
`-group_subset_at_random`	i f	Create a group of things from the existing group i by picking a random subset with probability f
`-create_regulus`	i N	Create a regulus for quadric i with N lines

Table 18: Scene definition commands (part 2)

Command	Arguments	Purpose
`-spheres`	i r prop	For each element in point group i, create a sphere of radius r with given Povray properties.
`-cylinders`	i r prop	For each element in edge group i, create a cylinder of radius r with given Povray properties.
`-prisms`	i d prop	For each element in face group i, create a prism of half-thickness d with given Povray properties.
`-planes`	i prop	For each element in plane group i, create a plane with given Povray properties.
`-lines`	i r prop	For each element in line group i, create a line of radius r with given Povray properties.
`-cubics`	i prop	For each element in group i of cubics, create a surface with given Povray properties.
`-quadrics`	i prop	For each element in group i of quadrics, create a surface with given Povray properties.
`-quartics`	i prop	For each element in group i of quartics, create a surface with given Povray properties.
`-texts`	i d s prop	For each element in group i of labels, create a text element with half-thickness d and size s with given Povray properties.

Table 19: Graphical output commands

edges are in red, and whose faces are yellow and transparent. The cube turns around a vertical axis of symmetry. Here is the first frame of the result:



The coordinates of the cube are stored in an object file `cube_centered.obj`. The content of this file is:

```
v -1 -1 -1
v 1 -1 -1
v -1 1 -1
v 1 1 -1
v -1 -1 1
v 1 -1 1
v -1 1 1
v 1 1 1
f 1 2 4 3
f 1 2 6 5
f 1 3 7 5
f 2 4 8 6
f 3 4 8 7
f 5 6 8 7
```

Here is a simple example of a cubic surface, called the monkey saddle. The equation of the surface is

$$z = x^3 - 3xy^2$$

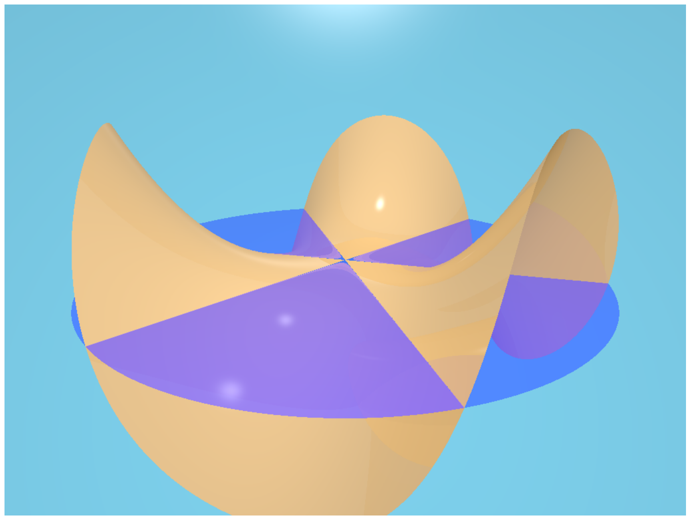
The example plots the surface together with the tangent plane at $(0, 0, 0)$, rotated around the z-axis.

```
      orbiter.out -v 2 -povray \
         -round 0 -nb_frames_default 30 -output_mask monkey_%d_%03d.pov \

         -video_options -W 1024 -H 768  -global_picture_scale 0.8 \
         -default_angle 75 -clipping_radius 0.8 \
         -camera 0 "0,0,1" "1,1,0.5" "0,0,0"  \
         -rotate_about_z_axis \
         -end \
         -scene_objects \
            -cubic_lex "1,0,0,0,-3,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,0" \
            -plane_by_dual_coordinates "0,0,1,0" \
            -group_of_things "0" \
            -group_of_things "0" \
            -cubics 0 "texture{ pigment{ Gold } finish {ambient 0.4 diffus
      e 0.5 roughness 0.001 reflection 0.1 specular .8} }" \
            -planes 1 "texture{ pigment{ color Blue transmit 0.5 } finish
      { diffuse 0.9 phong 0.2}}" \
         -scene_objects_end \
         -povray_end

```

Here is one of the frames that are created:



Here is another cubic surface, called Hilbert Cohn-Vossen. The equation of the surface is

$$\frac{5}{2}xyz - (x^2 + y^2 + z^2) + 1 = 0.$$

```
        orbiter.out -v 2 -povray \
           -round 0 -nb_frames_default 30 -output_mask HCV_%d_%03d.pov \
           -video_options -W 1024 -H 768  -global_picture_scale 0.9 \
           -default_angle 75 -clipping_radius 2.4 \
           -camera 0 "1,1,1" "-3,1,3" "0.12,0.12,0.12"  \
           -end \
           -scene_objects \
              -Hilbert_Cohn_Vossen_surface \
              -group_of_things "0" \
              -cubics 0 "texture{ pigment{ White*0.5 transmit 0.5 } finish
     {ambient 0.4 diffuse 0.5 roughness 0.001 reflection 0.1 specular
```

```
.8} }" \
            -group_of_things_as_interval 0 6 \
            -group_of_things_as_interval 6 6 \
            -group_of_things_as_interval_with_exceptions 12 15 "14,19,23" \
            -lines 1 0.02 "texture{ pigment{ color Red } finish { diffuse 0.9 phong 1}}" \
            -lines 2 0.02 "texture{ pigment{ color Blue } finish { diffuse 0.9 phong 1}}" \
            -lines 3 0.02 "texture{ pigment{ color Yellow } finish { diffuse 0.9 phong 1}}" \
            -label 0 "a1" -label 2 "a2" -label 4 "a3" \
            -label 6 "a4" -label 8 "a5" -label 10 "a6" \
            -label 12 "b1" -label 14 "b2" -label 16 "b3" \
            -label 18 "b4" -label 20 "b5" -label 22 "b6" \
            -label 24 "c12" -label 26 "c13" -label 30 "c15" \
            -label 32 "c16" -label 34 "c23" -label 36 "c24" \
            -label 40 "c26" -label 42 "c34" -label 44 "c35" \
            -label 48 "c45" -label 50 "c46" -label 52 "c56" \
            -group_of_things_as_interval 0 6 \
            -texts 4 0.2 0.15 "texture{ pigment{Black} } no_shadow" \
            -group_of_things_as_interval 6 6 \
            -texts 5 0.2 0.15 "texture{ pigment{Black} } no_shadow" \
            -group_of_things_as_interval 12 12 \
            -texts 6 0.2 0.15 "texture{ pigment{Black} } no_shadow" \
        -scene_objects_end \
        -povray_end

```

Figure 10 shows the final product. The Schlaefli labeling of lines can be seen. Orbiter can plot functions using a built-in function tracker. The functions must be continuous apart from a finite number of poles. The function can have multiple components, each described using an expression. Each expression is specified in Reverse Polish Notation (RPN). Consider an example. A Lissajous curve is defined using coordinate functions of the form

$$x = r\sin\big(at+c\big), \quad y = r\sin\big(bt\big), \quad a,b,c,r \in \mathbb{R}.$$

The terms

$$r\sin\big(at+c\big), \quad r\sin\big(bt\big)$$

are the expressions of the two coordinate functions. RPN means that the operator is listed after the operands. A stack data structure is used to hold temporary values. Operators are pushed to the top of the stack using the push commands. A binary operator pops the two elements from the stack, performs the operation, and pushes the resulting value back onto the stack. For a unary operator, only one element is poped and replaced by the result. Here

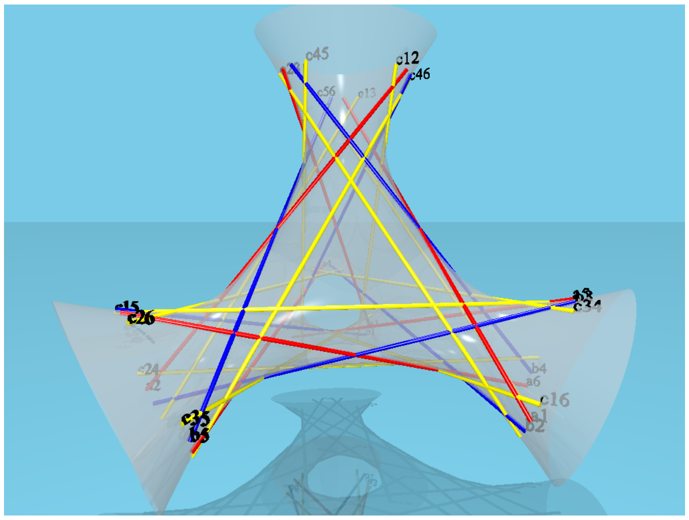


Figure 10: The Hilbert Cohn-Vossen surface

are some examples of expressions rewritten in RPN:

$$\begin{aligned} \sin(x) &\mapsto \text{ push } x \ \text{ sin}, \\ a+b &\mapsto \text{ push } a \text{ push } b \text{ add}, \\ a\cdot b &\mapsto \text{ push } a \text{ push } b \text{ mult}. \end{aligned}$$

The coordinate functions are enclosed between `-code` and `-code_end` commands. Each coordinate function is described in RPN and terminated using a `return` keyword. By the time the `return` keyword is reached, the RPN expression must have exactly one value on the stack which is considered the value of the expression. Constants are declared between the `-const` and `-const_end` keywords. Likewise, variables are declared between the `-var` and `-var_end` keywords. Picking $a = 3$, $b = 2$, $c = \pi/2$ and $r = 7$, the function is computed using

```
orbiter.out -v 2 -smooth_curve "lissajous" 0.07 2000 15 0 18.85 \
    -const a 3 b 2 c 1.57 r 7 -const_end \
    -var t -var_end \
    -code \
      push t push a mult push c add sin push r mult return \
      push t push b mult sin push r mult  return \
    -code_end
```

The sequence

```
push t push a mult push c add sin push r mult
```

is $r\sin(at + c)$ expressed in RPN. The constants are defined in the line

```
-const a 3 b 2 c 1.57 r 7 -const_end
```

The input variable is defined using the line

```
-var t -var_end
```

The sequence

```
-smooth_curve "lissajous" 0.07 2000 15 0 18.85
```

defines the name of the output file, the fact that two consequtive points are never further than $\epsilon = 0.07$ away, the fact that points that are 15 or more away from the origin should be ignored, and the fact that the variable t loops over the range $[0, 18.85]$ with a default of 2000 steps. The evaluator automatically reduces the step-size if consecutive image points are more than ϵ apart. The code to produce the plot is

```
orbiter.out -v 2 -povray \
   -round 0 -nb_frames_default 1 -output_mask lissajous_%d_%03d.pov \
   -video_options -W 1024 -H 768  -global_picture_scale 0.40 \
   -default_angle 45 -clipping_radius 5 -omit_bottom_plane  \
   -camera 0 "0,-1,0" "0,0,12" "0,0,0"  \
   -rotate_about_z_axis \
   -end \
   -scene_objects \
      -line_through_two_points_recentered_from_csv_file coordinate_grid.csv \
      -group_of_things "0" \
      -group_of_things "1" \
      -group_of_things "2" \
      -lines 0 0.09 "texture{ pigment{ color Yellow } }" \
      -lines 1 0.09 "texture{ pigment{ color Yellow } }" \
      -lines 2 0.09 "texture{ pigment{ color Yellow } }" \
      -group_of_things_as_interval 3 39 \
      -lines 3 0.02 "texture{ pigment{ color Black } }" \
      -point_list_from_csv_file function_lissajous_N2000_points.csv \
      -group_of_things_as_interval 0 6524\
      -spheres 4 0.1 "texture{ pigment{ color Red } finish { diffuse 0.9 phong 1}}" \
      -plane_by_dual_coordinates "0,0,1,0" \
      -group_of_things "0" \
      -planes 5 "texture{ pigment{ color Blue*0.5 transmit 0.5 } }" \
   -scene_objects_end \
   -povray_end

```

The plot is shown in Figure 11. We can turn it into a 3D plot by using the t value for the z coordinate. The code to produce the 3D plot is

```
orbiter.out -v 2 -povray \
   -round 0 -nb_frames_default 30 -output_mask lissajous_3d_%d_%03d
```

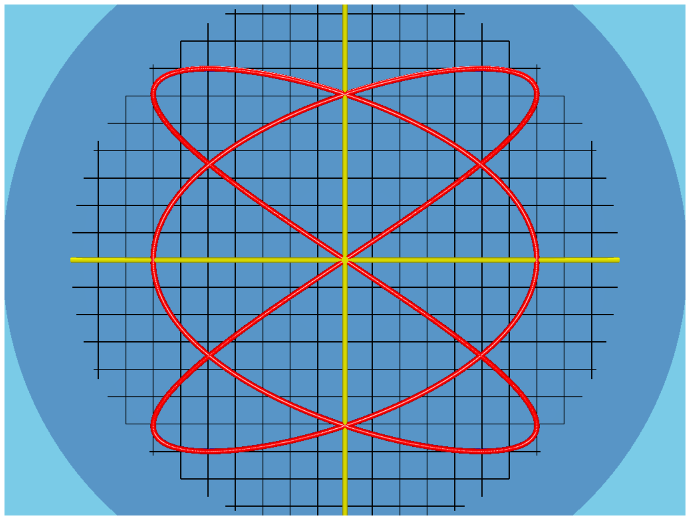


Figure 11: Lissajous figure

```
.pov \
   -video_options -W 1024 -H 768  -global_picture_scale 0.40 \
   -default_angle 45 -clipping_radius 5 -omit_bottom_plane  \
   -camera 0 "0,0,1" "7,7,5" "0,0,1"  \
   -rotate_about_z_axis \
   -end \
   -scene_objects \
      -line_through_two_points_recentered_from_csv_file coordinate_g
rid.csv \
      -group_of_things "0" \
      -group_of_things "1" \
      -group_of_things "2" \
      -lines 0 0.09 "texture{ pigment{ color Yellow } }" \
      -lines 1 0.09 "texture{ pigment{ color Yellow } }" \
      -lines 2 0.09 "texture{ pigment{ color Yellow } }" \
      -group_of_things_as_interval 3 39 \
      -lines 3 0.02 "texture{ pigment{ color Black } }" \
      -point_list_from_csv_file function_lissajous_3d_N2000_points.c
sv \
      -group_of_things_as_interval 0 6538\
      -spheres 4 0.1 "texture{ pigment{ color Red } finish { diffuse
 0.9 phong 1}}" \
      -plane_by_dual_coordinates "0,0,1,0" \
      -group_of_things "0" \
   -scene_objects_end \
   -povray_end
```

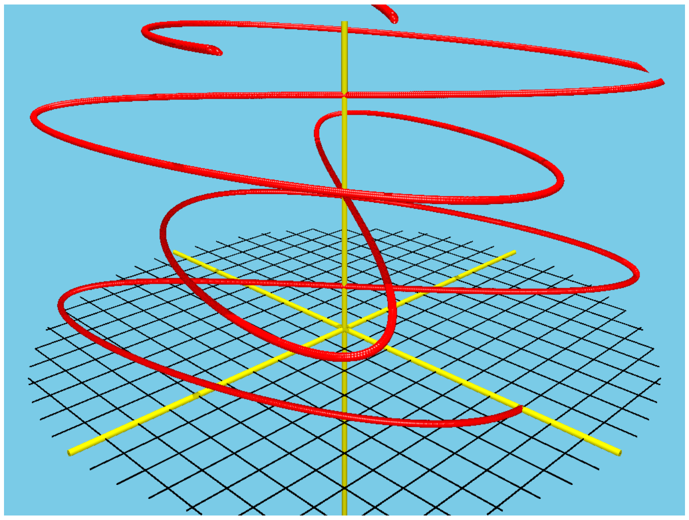


Figure 12: Lissajous Spacecurve

The function is computed using the command

```
orbiter.out -v 2 -smooth_curve "lissajous_3d" 0.07 2000 50 0 18.85 \
    -const a 3 b 2 c 1.57 r 7 -const_end \
    -var t -var_end \
    -code \
      push t push a mult push c add sin push r mult return \
      push t push b mult sin push r mult  return \
      push t return \
    -code_end \
```

The 3D curve is shown in Figure 12.

14 Cryptography and Number Theory

In Table 20, some number theoretic commands are shown. For instance,

```
orbiter.out -v 2 -inverse_mod  18059241 58014043
```

conputes the inverse of 18059241 modulo 58014043. The command

```
orbiter.out -v 5 -jacobi 2221 7817
```

computes the Jacobi symbol

$$\left(\frac{2221}{7817}\right).$$

The denominator p has to be a positive odd integer.

In Table 21, some cryptographic commands are shown. For instance,

Command	Arguments	Description
`-trace`	q	Computes the absolute trace function for all elements in $\mathbb{F}_q$
`-norm`	q	Computes the absolute norm function for all elements in $\mathbb{F}_q$
`-jacobi`	$a\ p$	Computes the Jacobi symbol $\left(\frac{a}{p}\right)$
`-power_mod`	$a\ n\ p$	Raises a to the power n modulo p
`-primitive_ root`	p	Computes a primitive root modulo p
`-discrete_log`	$b\ a\ p$	Computes n such that $a^n \equiv n \mod p$
`-square_ root_mod`	$a\ p$	computes a square root of a modulo p
`-inverse_mod`	$a\ p$	computes the modular inverse of a modulo p
`-sift_smooth`	$a\ n$ primes	Computes all smooth numbers in the interval $[a, a+n-1]$. Smooth means that they factor completely over the list of primes given.
`-solovay_ strassen`	$a\ n$	Performs n Solovay / Strassen tests on the number a
`-miller_rabin`	$a\ n$	Performs n Miller / Rabin tests on the number a
`-fermat`	$a\ n$	Performs n Fermat tests on the number a
`-find_pseudoprime`	$a\ n_1\ n_2\ n_3$	Computes a pseudoprime which survives n_1 Fermat tests, n_2 Miller Rabin tests, n_3 Solovay Strassen tests
`-find_strong_ pseudoprime`	$a\ n_1\ n_2$	Computes a pseudoprime which survives n_1 Fermat tests and n_2 Miller Rabin tests
`-random`	n fname	Creates n random numbers and writes them to the csv file `fname`
`-random_last`	n	Creates n random numbers prints the last one
`-affine_sequence`	$a\ b\ p$	Splits the interval $[0, p-1]$ into affine sequences of the form $x_{n+1} = ax_n + b \mod p$

Table 20: Number Theoretic Commmands

Command	Arguments	Description
`-RSA_encrypt_text`	d n b text	Using blocks of b letters at a time, encrypt "text" using RSA with exponent d modulo n
`-RSA`	d n list-of-integers	encrypt the given sequence of integers using RSA with exponent d modulo n
`-EC_add`	p a b i_1 i_2	On the elliptic curve $y^2 \equiv x^3 + ax + b \mod p$, add the points with ibdices i_1 and i_2, each given as a pair x, y
`-EC_points`	p a b	Computes all points over $\mathbb{F}_p$ of the elliptic curve $y^2 \equiv x^3 + ax + b \mod p$
`-EC_multiple_of`	p a b pt n	Computes the n fold multiple of the given point pt on the elliptic curve $y^2 \equiv x^3 + ax + b \mod p$
`-EC_cyclic_ subgroup`	p a b pt	Computes the cyclic subgroup generated by the given point pt on the elliptic curve $y^2 \equiv x^3 + ax + b \mod p$
`-EC_Koblitz_ encoding`	p a b s pt plain	Computes the Koblitz encoding of "plain" (all caps) on the elliptic curve $y^2 \equiv x^3 + ax + b \mod p$ using the base point pt and the secret exponent s
`-EC_bsgs`	p a b pt n cipher	Prepare the baby-step giant-step tables for the ciphertext "cipher" on the elliptic curve $y^2 \equiv x^3 + ax + b \mod p$ using the base point pt of order n
`-EC_bsgs_decode`	p a b pt n cipher round-keys	Decodes the ciphertext "cipher" on the elliptic curve $y^2 \equiv x^3 + ax + b \mod p$ using the base point pt of order n and the round keys "keys"
`-EC_discrete_log`	p a b pt base-pt	Computes the elliptic curve discrete log analogue of pt with respect to base-pt on the elliptic curve $y^2 \equiv x^3 + ax + b \mod p$

Table 21: Cryptographic Commmands

```
orbiter.out -v 2 -EC_add 11 1 3 "1,4" "1,4"
```

adds the point $(1,4)$ on the curve $y^2 = x^3 + x + 3 \mod 11$ to itself. The command

```
orbiter.out -v 2 -EC_cyclic_subgroup 11 1 3 "1,4"
```

computes the cyclic subgroup generated by the point $(1,4)$ on the curve $y^2 = x^3 + x + 3 \mod 11$. The command

```
orbiter.out -v 2 -EC_points 199 5 7
```

computes all points on the curve $y^2 = x^3 + 5x + 7 \mod 199$. The command

```
orbiter.out -v 6 -seed 17 -EC_Koblitz_encoding 199 5 7 67 "147,164" "DEADBEEF"
```

encode the message "DEADBEEF" on the curve $y^2 = x^3 + 5x + 7 \mod 199$ using the base point $(147,164)$ and the secret key 67. The ith input character is encoded as two points (R_i, T_i) on the curve using the Elgamal scheme. A random round key is generated for each plaintext symbol. As seen in this example, the `-seed` command can be used to seed the random number generator with an arbitrary integer (here 17). The command

```
orbiter.out -v 2 -EC_bsgs 199 5 7 "147,164" 212 \
  "172,158,45,195,50,22,10,103,55,33,50,22,145,105,31,74,73,155,67,60,25,6"
```

performs a baby-step-giant-step brute force attack on the ciphertext sequence

$$R_i = (172,158),(45,195),(50,22),(10,103),(55,33),\\ (50,22),(145,105),(31,74),(73,155),(67,60),(25,6),$$

using the base point $(147,164)$ on the curve $y^2 = x^3 + 5x + 7 \mod 199$, assuming a group order of 212. The command

```
orbiter.out -v 2 -EC_bsgs_decode 199 5 7 "129,176" 212 \
  "127,188,51,141,85,29,106,90,41,105,179,71,171,2,16,197,183,72,27,129,37,10" \
  "50,179,169,13,153,169,115,116,188,110,176"
```

performs a decoding of the ciphertext sequence

$$T_i = (127,188),(51,141),(85,29),(106,90),(41,105),(179,71),\\ (171,2),(16,197),(183,72),(27,129),(37,10),$$

assuming round keys

$$k_i = 50,179,169,13,153,169,115,116,188,110,176,$$

using the base point $(147,164)$ on the curve $y^2 = x^3 + 5x + 7 \mod 199$, and assuming a group order of 212.

Command	Arguments	Description
`-RREF`	q m n list-of-integers	Compute the RREF of the $m \times n$ matrix over $\mathbb{F}_q$
`-nullspace`	q m n list-of-integers	Compute a basis for the right nullspace of the given $m \times n$ matrix
`-normalize_from_the_right`		Normalizes the result of `-RREF` or `nullspace` from the right
`-weight_enumerator`	q m n list-of-integers	Computes the weight enumerator of the linear code generated by the given $m \times n$ matrix
`-BCH`	n q t	Creates the BCH-code of length n over the field $\mathbb{F}_q$ with designed distance t

Table 22: Coding Theoretic Commmands

15 Coding Theory

Orbiter can classify linear codes with bounded minimum distance. To do so, Orbiter establishes a certain matroid in a suitable projective geometry and computes the orbits under the projective group. Recall that a linear $[n,k]$-code $\mathcal{C}$ over $\mathbb{F}_q$ is a k-dimensional subspace of $\mathbb{F}_q^n$. The code is said to have minimum distance d if

$$\min_{\substack{c,c' \in \mathcal{C} \\ c \neq c'}} d(c,c') = d$$

where $d(\mathbf{x},\mathbf{y})$ is the Hamming matric on $\mathbb{F}_q^n$, which counts the number of positions where $\mathbf{x}$ and $\mathbf{y}$ differ. A code code has both k and d large with respect to n. The notion of isometry with respect to the Hamming metric leads to a notion of equivalence of codes. Two codes are equivalent if the coordinates of the vectors in one code can be computed (simultaneously) so as to obtain the second code. The automorphism group is the set of isometry maps from one code to itself.

In Table 22, some coding theoretic commands of Orbiter are shown.

The classification problem of optimal codes in coding theory is the problem of determining the equivalence classes of codes for a given set of values of n and k with a lower bound on d. We wish to use Orbiter for solving this problem for small instances.

Orbiter reduces the problem of classifying $[n,k,\geq d]$ codes over $\mathbb{F}_q$ to an equivalent problem in finite geometry. Acccording to [2], the equivalence classes of $[n,k,\geq d]$ codes over $\mathbb{F}_q$ for $d \geq 3$ are in canonical one-to-one correspondence to the sets of size n in $\mathrm{PG}(n,k-1,q)$ with the property that any set of size at most $d-1$ is linearly dependent. Let $\Lambda_{m,s}(q)$ be the the poset of subsets of $\mathrm{PG}(m,q)$ such that any set of s or less points is independent. The group $G = \mathrm{P\Gamma L}(m+1,q)$ acts on this poset. For $m = n-k-1$ and $s = d-1$, the orbits of G on

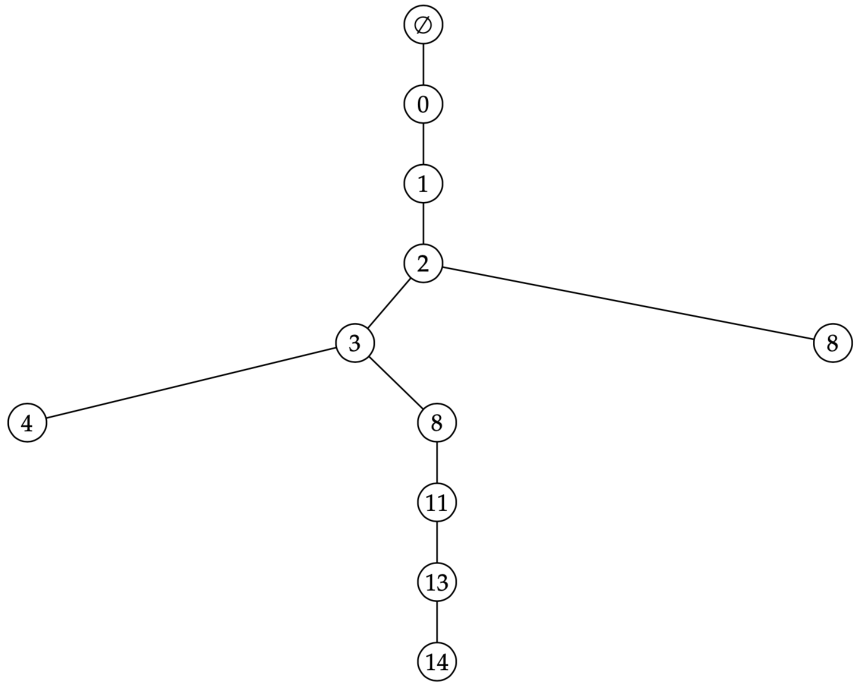


Figure 13: Orbits of PGL$(4,2)$ on the poset $\Lambda_{3,3}(2)$,

sets in $\Lambda_{m,s}(q)$ of size n are in canonical one-to-one correspondence to the $[n,k,\geq d]$ codes over $\mathbb{F}_q$.

The Orbiter command

```
orbiter.out -v 2 -code_classify -n 8 -k 4 -q 2 -d 4 -lex
```

can be used to classify the $[8,4,4]$ codes over $\mathbb{F}_2$. It turns out that there is exactly one such code, the $[8,4,4]$ extended Hamming code. Using the group PGL$(4,2)$ acting on the poset $\Lambda_{3,3}(2)$, Orbiter produced the poset of orbits shown in Figure 13. In this diagram, the numbers stand for Orbiter numbers of points in PG$(3,2)$. All nodes except for the root node have a number attached to it. The nodes represent subsets. In order to determine the set associated to a node, follow the path from the root node to the node and collect the points according to their labels. The root node represents the empty set. The $[8,4,4]$-code is represented by the set $\{0,1,2,3,8,11,13,14\}$. The fact that there is only one node at level 8 in the poset of orbits tells us that the code is unique up to equivalence. Let us look at the code. The elements of the set $\{0,1,2,3,8,11,13,14\}$ are points in PG$(3,2)$. We write the coordinate vectors in the columns of a matrix H like so:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

This matrix is the parity check matrix H of the code $\mathcal{C}$. This means that the words of the code are the vectors c such that $c \cdot H^{\top} = 0$. Observe that the vectors that we put in the columns of H all have odd weight. They are in fact the points of the hyperplane $x+y+z+w=0$. This shows that the stabilizer of the code which is the stabilizer of the set is equal to $\mathrm{AGL}(3,2)$, a group of order 1344.

References

[1] Susan Barwick and Gary Ebert. *Unitals in projective planes.* Springer Monographs in Mathematics. Springer, New York, 2008.

[2] Anton Betten, Michael Braun, Harald Fripertinger, Adalbert Kerber, Axel Kohnert, and Alfred Wassermann. *Error-correcting linear codes*, volume 18 of *Algorithms and Computation in Mathematics.* Springer-Verlag, Berlin, 2006. Classification by isometry and applications, With 1 CD-ROM (Windows and Linux).

[3] Anton Betten, James W. P. Hirschfeld, and Fatma Karaoğlu. Classification of cubic surfaces with twenty-seven lines over the finite field of order thirteen. *Eur. J. Math.*, 4(1):37–50, 2018.

[4] Anton Betten and Fatma Karaoğlu. Cubic surfaces over small finite fields. *Des. Codes Cryptogr.*, 87(4):931–953, 2019.

[5] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[6] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.8.7*, 2017.

[7] D. G. Glynn and J. W. P. Hirschfeld. On the classification of geometric codes by polynomial functions. *Des. Codes Cryptogr.*, 6(3):189–204, 1995.

[8] Fatma Karaoğlu. The cubic surfaces with twenty-seven lines over finite fields, Ph.D. thesis, University of Sussex, 2018.

[9] Wen-Ch'ing Winnie Li. Character sums and abelian Ramanujan graphs. *J. Number Theory*, 41(2):199–217, 1992. With an appendix by Ke Qin Feng and the author.

[10] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[11] Nauty User's Guide (Version 2.6), Brendan McKay, Australian National University, 2016.

[12] POV-RAY Developers. POV-RAY, Persistence of Vision Raytracer Pty. Ltd. (2003-2008) `http://povray.org`, accessed 4/2/2017.

[13] L. Schläfli. An attempt to determine the twenty-seven lines upon a surface of the third order and to divide such surfaces into species in reference to the reality of the lines upon the surface, *Quart. J. Math.* **2** (1858), 55–110.

[14] Magma system. Magma Calculator `http://magma.maths.usyd.edu.au/calc/`, accessed 11/24/2019.

[15] Robert Wilson, Peter Walsh, Jonathan Tripp, Ibrahim Suleiman, Richard Parker, Simon Norton, Simon Nickerson, Steve Linton, John Bray, and Rachel Abbott. ATLAS of Finite Group Representations - Version 3, `http://brauer.maths.qmul.ac.uk/Atlas/v3/`, accessed 11/24/2019.