Results of computer search for a perfect cuboid

Robert D. Matson

Abstract

A suite of optimized computer programs was designed to systematically search for a perfect cuboid, keep track of close misses, and investigate the statistical trend of these near matches with increasing Euler brick size. While no perfect cuboid was found, the minimum length of the odd side has been extended from the prior published limit of 3 trillion \(^3\) \((3 \times 10^{12})\) to 8 trillion \((8 \times 10^{12})\).

Background

The Euler brick is named after Leonhard Euler, and is a rectangular parallelepiped (or cuboid) which has integer dimensions for not only its length, width and height, but also its three face diagonals. The smallest Euler brick was discovered by Paul Halcke in 1719 and has edge dimensions of 44, 117 and 240, and face diagonals of 125, 244 and 267. A primitive Euler brick is an Euler brick whose edge lengths are relatively prime. Note that there are an infinite number of primitive Euler bricks, and that all primitive Euler bricks have one odd edge length, and two even edge lengths.

A perfect cuboid is an Euler brick whose space diagonal also has integer length. Stated mathematically, if there is a perfect cuboid it must satisfy the following set of Diophantine equations:

\[
\begin{align*}
  a^2 + b^2 &= d^2 \\
  a^2 + c^2 &= e^2 \\
  b^2 + c^2 &= f^2 \\
  a^2 + b^2 + c^2 &= g^2
\end{align*}
\]

As of April 2014, no example of a perfect cuboid has been found, but no one has proven that none exist. Past exhaustive computer searches have determined that the smallest edge must be
greater than $10^{10}$ [R. Rathbun$^2$], and the odd edge must be greater than $3 \times 10^{12}$ [B. Butler$^1$]. The main result from this paper is that the odd edge is now proven to be greater than $8 \times 10^{12}$.

Some interesting facts are known about the dimensions of a primitive perfect cuboid (assuming one exists) based on modular arithmetic. As mentioned above, one side must be odd and the other two sides must be even. Additionally:

- Two edges must be divisible by 3, and at least one of those edges must be divisible by 9.
- Two edges must be divisible by 4, and one of those must be divisible by 16.
- One edge is divisible by 5.
- One edge is divisible by 7. [T. Roberts$^3$]
- One edge is divisible by 11.
- One edge is divisible by 19. [T. Roberts$^3$]

Bill Butler’s algorithm took advantage of the divisibility-by-16 requirement; the author’s algorithm exploits this feature as needed, but more often uses the more restrictive divisibility-by-19 requirement.

**Algorithmic Approach**

The core of the algorithmic approach is the same as that pioneered by Butler, but with some enhancements to increase the search speed while running with essentially unlimited precision. (Butler’s program carried 64-bit precision, but was designed in such a way to err on the side of generating a false positive rather than skipping over a real solution. Even so, his program did not generate any false positives. Since I wished to keep track of near-misses, my algorithm supports up to 180 bits of precision.)

Butler’s approach takes advantage of the fact that all Pythagorean triples $<X,Y,Z>$ (where $X$ is the odd side, $Y$ is the even side, and $Z$ is the hypotenuse) obey the following set of equations (Euclid’s Formula):

\[
\begin{align*}
X &= (P^2 - Q^2) \times K = (P - Q) \times (P + Q) \times K \\
Y &= 2 \times P \times Q \times K \\
Z &= (P^2 + Q^2) \times K
\end{align*}
\]

where $P$, $Q$ and $K$ are integers. Since $X$ is odd, $K$ must be odd, and either $P$ or $Q$ must be odd (but not both). Butler’s algorithm finds all possible values for the even side, $Y$, based on all possible integer triplet products $<D1*D2*D3>$ that equal $X$:
\[ D_1 = P - Q \]
\[ D_2 = P + Q \quad \text{(note D2 always greater than D1)} \]
\[ D_3 = K \]

Solving for \( P, Q \) and \( Y \):

\[ P = \frac{D_1 + D_2}{2} \]
\[ Q = \frac{D_2 - D_1}{2} \]
\[ Y = \frac{(D_1 + D_2) \ast (D_2 - D_1) \ast D_3}{2} = \frac{(D_2^2 - D_1^2) \ast D_3}{2} \]

Substituting \( \frac{X}{D_1 \ast D_3} \) for \( D_2 \), and simplifying:

\[ Y = \frac{X^2 - (D_1^2 \ast D_3)^2}{2 \ast D_1^2 \ast D_3} \quad \frac{X^2 - i^2}{2i} \quad (1) \]

The form of this final expression will be useful later since it indicates how large \( Y \) can be for a given \( X \).

For each \( X \), all possible \( Y \) values are stored in a list. The more prime factors that \( X \) has, the greater the number of possible \( Y \) edges. (For \( X \) values over 4 trillion, the number of \( Y \) edges can exceed 100,000.) In Butler’s algorithm, this \( Y \) candidate list is further subdivided into those that are divisible by 16 and those that aren’t. As discussed earlier, at least one side must come from the divisible-by-16 group, so this subdivision eliminates a lot of potential combinations, speeding up processing. This methodology is also used in the author’s program, but only about 5% of the time. If \( X \) is not divisible by 19 (which happens 94.7% of the time), then one of the sides must be. So in these cases, instead of keeping track of \( Y \)s that are divisible by 16, the code subdivides the list based on divisibility by 19. (Note that this is much more restrictive than divisibility by 16 since all \( Y \)s are known to be divisible by 4.) The \( Y \)s are divided by 4, squared, and stored in a separate \( Y^2/16 \) list.

The final step is to compute the sums of all possible pairs of values in the \( Y^2/16 \) lists, where one pair member is always from the divisibility by 19 (or 16) group. The result of this addition is then checked against all other members in the \( Y^2/16 \) list for a match. The following graphic helps illustrate why this works:
If a perfect cuboid exists, then \( Y_1 \) and \( Y_2 \) will both be in the \( Y \) edge list, but so will \( Y_3 \). So if the sum of any pair of values in the \( Y^{2/16} \) list can be found elsewhere in that same list, then a perfect cuboid has been found.

**Hashing**

The \( Y \) lists can get quite large when the \( X \) values are the products of many small primes, requiring the number of \( Y^{2/16} \) comparisons to grow in proportion to \( Y^3 \). Butler devised a clever system for greatly decreasing the number of required comparisons by building a 14-bit (0-16383) hash table of the \( Y^{2/16} \) values. For each pair of \( Y \) values, if the sum of their hash values cannot be found in the hash table, then that combination can be skipped. If the hash sum *is* found in the hash table, then it is still only necessary to check those few cases that share the target hash value.

**Eliminating the High-Y Solutions**

Precision and to some extent computational speed are driven by how large the \( Y \) values can become. Examining equation (1), it is apparent that maximum \( Y \) occurs when the \( <D_1, D_2, D_3> \) triplet product for \( X \) is \( <1, X, 1> \). In this case, \( i=1 \), and
\[ Y_1 = \frac{X^2 - 1}{2} \] \hspace{1cm} (2)

So if \( X \) values up to \( 10^{13} \) are to be checked, \( Y \) could be as large as \( 5 \times 10^{25} \) (86 bits). Since the algorithm must compare the square of one candidate \( Y \) to the sum of the squares of two other candidate \( Y \)s, carrying full precision would appear to require 172 bits. However, all candidate \( Y \)s are divisible by 4, and therefore all candidate \( Y^2 \) values are divisible by 16, so the lowest four bits are always zero. So only 168 bits are needed to test up to 10 trillion. Butler’s algorithm uses the C function \textit{hypot}, and flags any case where the computed hypotenuse differs from a target \( Y \) value by less than one part in \( 10^{12} \) (i.e. 40 bits of precision). While Butler’s algorithm is not too concerned with how large \( Y \) and \( Y^2 \) can be, the full-precision calculation would benefit if the largest \( Y \) values could somehow be ruled out as potential solutions. Fortunately, they can.

After \( <1, X, 1> \), the next two highest possible solutions for \( Y \) are based on the triplets \( <1,X/3,3> \) and \( <1, X/5, 5> \). Equation (2) can only be a solution if it is the hypotenuse of two smaller solutions to equation (1). But even the sum of the next two smaller solutions is not as large as \( (X^2-1)/2 \):

\[
<1, X/3, 3>: \quad Y_3 = \frac{(X^2 - 9)}{6}
\]

\[
<1, X/5, 5>: \quad Y_5 = \frac{(X^2 - 25)}{10}
\]

\[
Y_3 + Y_5 = \frac{(8 \times X^2 - 120)}{30} = \frac{(4/15)X^2 - 40}{30}
\]

Since the sum of any other pair of \( Y \) values is always less than \( Y_1 \), \( Y_1 \) cannot be a solution. Similar logic can eliminate \( Y_3 \) as a solution:

\[
Y_7 = \frac{(X^2 - 49)}{14}
\]

\[
Y_5^2 + Y_7^2 = \frac{(X^2-25)^2}{100} + \frac{(X^2-49)^2}{196} = \frac{[49 \times (X^2 - 25)^2 + 25 \times (X^2 - 49)^2]}{4900}
= \frac{(74 \times X^4 - 4900 \times X^2 + 90650)}{4900} = \frac{(37/2450) \times X^4 - X^2 + 18.5}{4900}
\]

\[
Y_3^2 = \frac{(1/36) \times X^4 - 0.5 \times X^2 + 2.25}{4900}
\]

Since we know \( X \) must be large, only the \( X^4 \) term matters; the \( X^2 \) and smaller terms can be ignored. Since \( 1/36 \) is greater than \( 37/2450 \), the sum of the squares of \( Y_5 \) and \( Y_7 \) will always be less than \( Y_3 \). Therefore, \( Y_3 \) cannot be a solution.

The same procedure can be used to eliminate \( Y_5 \) as a solution. \( Y_7 \) is the first case where the sum of the squares of the next two smaller \( Y \)s exceeds \( Y_7^2 \):
\[ Y_7^2 \approx X^4 / 196 \approx 0.00510 \times X^4 \]

\[ Y_9^2 \approx X^4 / 324 \quad \text{and} \quad Y_{11}^2 \approx X^4 / 484 \]

\[ Y_9^2 + Y_{11}^2 \approx 0.00515 \times X^4 \]

So \( Y_7 \) cannot be immediately ruled out as a potential solution.

A separate program was written to systematically eliminate the highest \( Y \) solutions. Per equation (1), all \( Y \)s are of the form:

\[
Y_i = \frac{X^2 - i^2}{2i}
\]

For there to be a perfect cuboid solution, there must be a set of three odd integers \( A, B \) & \( C \) such that:

\[
\frac{(X^2 - A^2)^2}{4A^2} = \frac{(X^2 - B^2)^2}{4B^2} + \frac{(X^2 - C^2)^2}{4C^2}
\]

or more simply:

\[
\frac{(X^2 - A^2)^2}{A^2} = \frac{(X^2 - B^2)^2}{B^2} + \frac{(X^2 - C^2)^2}{C^2}
\]

The quadratic solution for \( X^2 \) is:

\[
X^2 = \frac{K2 \pm ABC \times \text{SQRT}(K2 - K1*K3)}{K1}
\]

where

\[
K1 = A^2*(B^2 + C^2) - B^2*C^2
\]

\[
K2 = A^2*B^2*C^2
\]

\[
K3 = B^2 + C^2 - A^2
\]

\( A < B < C \)
For any given A, there is a finite set of odd values of B and C that produce real solutions for $X^2$, and the vast majority of these solutions will not be integers. If no integer solutions are found, $Y_A$ cannot be the even side of any Pythagorean triple.

Occasionally, an integer $X^2$ solution will exist for a given value of A. The lowest A for which this happens is $A=329$, $B=399$, $C=423$:

$$\frac{(X^2 - 329^2)^2}{108241} + \frac{(X^2 - 399^2)^2}{159201} + \frac{(X^2 - 423^2)^2}{178929} = \frac{108241}{108241} + \frac{159201}{159201} + \frac{178929}{178929}$$

$$X^2 = 618849 = 3^2 \cdot 7 \cdot 11 \cdot 19 \cdot 47$$

However, it is not enough that $X^2$ be an integer; it must also be a perfect square, which 618849 is not. So $Y_{329}$ is not a possible solution.

All values of A were tested up through 5001, and none were found to produce integer solutions for $X$. Table 1 below lists all the primitive A-B-C combinations that produced integer solutions for $X^2$:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>$X^2$</th>
<th>$X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>329</td>
<td>399</td>
<td>423</td>
<td>618849</td>
<td>786.6696</td>
</tr>
<tr>
<td>473</td>
<td>507</td>
<td>1287</td>
<td>81295929</td>
<td>9016.4255</td>
</tr>
<tr>
<td>1881</td>
<td>1889</td>
<td>17193</td>
<td>1851221889</td>
<td>43025.8282</td>
</tr>
<tr>
<td>1953</td>
<td>2337</td>
<td>3317</td>
<td>162788409</td>
<td>12758.8561</td>
</tr>
<tr>
<td>2165</td>
<td>2167</td>
<td>10835</td>
<td>96646033</td>
<td>9830.8714</td>
</tr>
<tr>
<td>2185</td>
<td>2189</td>
<td>4577</td>
<td>23914825</td>
<td>4890.2786</td>
</tr>
<tr>
<td>2185</td>
<td>2189</td>
<td>5225</td>
<td>23914825</td>
<td>4890.2786</td>
</tr>
<tr>
<td>3385</td>
<td>4303</td>
<td>4615</td>
<td>15910177</td>
<td>3988.7563</td>
</tr>
<tr>
<td>4617</td>
<td>4647</td>
<td>36727</td>
<td>13824299889</td>
<td>117576.783</td>
</tr>
</tbody>
</table>

Table 1. All integer solutions for $X^2$ for $A < 5000$.

What this means is that there are no solutions to equation (1) where $D1^2 \cdot D3$ is less than 5000. So the largest that $Y$ can be is:

$$Y = \frac{X^2 - (5000)^2}{10000}$$
If values of X are to be tested up to $10^{13}$, Y can be as large as $10^{22}$, and $Y^2/16$ can be as large as $6.25 \times 10^{42}$ (143 bits). This is a significant decrease from the 168 bits required if the highest Y values had not been mathematically eliminated.

**Results**

The main program has been running on several cores of a Dell XPS 8500 3.40-GHz 64-bit machine for approximately 3 years, with no perfect cuboid solutions found for the odd side, X, less than 8 trillion. A second program has been tallying “close matches” for the last two months in order to investigate trends with increasing X. By far the closest match found (in terms of percentage of matching bits) was for one of the smallest odd sides, $X = 25025$:

\[
\begin{align*}
\text{Edge}_1 & : 71820 \\
\text{Edge}_2 & : 5088 \\
\text{Diagonal} & : 72000
\end{align*}
\]

\[
\text{Edge}_1^2 + \text{Edge}_2^2 = 5184000144
\]

\[
\text{Diagonal}^2 = 5184000000
\]

So the mismatch is only 144 out of 5184000000, or 0.0000028%. How close is the space diagonal to being an integer?

\[
\text{SQRT} \left( 25025^2 + 71820^2 + 5088^2 \right) \approx 76225.00094
\]

The close-match program keeps track of the number of high-order consecutive bits that match between the sum of the squares of the two sides divided by 16, and the square of the diagonal divided by 16 (remembering that the low order 4 bits will always be zero). For instance, in the above case:

\[
\frac{(\text{Edge}_1^2 + \text{Edge}_2^2)}{16} = 324000000 = \text{1001101001111101100100001001}
\]

\[
\frac{\text{Diagonal}^2}{16} = 324000000 = \text{1001101001111101100100000000}
\]

The highest 25 bits match out of a total of 29, or 86.2%. I arbitrarily chose to report any case where greater than 60% of the high-order bits matched. A plot of the results of all such cases up to $X = 200$ billion is shown below:
The red diamond corresponds to the X=25025 case – the only instance of greater than 80% of bits matching. A perhaps more telling trend is the plot of least-significant matching bit (where all higher bits match). As X increases, the point of $Y^2$ mismatch (reading from most-significant to least-significant bit) is rising at nearly the same rate as the number of bits in X. This trend suggests that it is highly unlikely that there is a perfect cuboid:
One case worth mentioning is $X=117,348,114,345$:

Edge1 = 9593 20475 90764
Edge1\(^2\) / 16 = 57 51848 59684 78806 10511 31481

Edge2 = 3 64478 66756 12448
Edge2\(^2\) / 16 = 83027 93694 20127 51527 72432 84544

Diagonal = 3 64604 89393 96864

Diagonal\(^2\) / 16 = 83085 45542 79812 30333 82944 16025
These differ by $38601369$, which is less than $1$ part in $2 \times 10^{22}$. The hypotenuse for Edge1 and Edge2 is $\approx 36460489396864.0000000847$. The reason Butler’s program did not flag this case is that his hash values are computed from bits 18-31; when the hashes do not match (as is the case here), then the algorithm ignores the case.

**Summary**

There are no perfect cuboids with odd side less than 8 trillion. While some remarkable near-misses have been identified, the overall trend with increasing odd side does not favor the existence of a perfect cuboid.

**References**

