Table S1. Percentage of the four nucleotides found at the three codon positions. Figures calculated by summing across all non-overlapping genes in all viral genomes examined.

Nucleotide	First codon position	Second codon position	Third codon position
A	30.6	31.2	26.0
G	30.9	17.0	22.6
C	18.6	22.9	21.8
T	19.8	28.9	29.6

Table S2. As Table 2 in main text, but we have excluded all genes that contain potentially misleading features such as multiple overlaps, internal frameshifts, splicing, or reverse complementation. (The cutoff for homology is 10^{-3} as in Table 2.)

Overlap Type	Gene Type	Number	Mean PDI	Number Nucleocapsid or Replicase	Number +1/-1 frameshift
Internal Overlap	Primary	31	6.4^{***}	14^{***}	$24/7^{**}$
	Secondary		1.7^{*}	0^{*}	
Terminal Overlap	3' Overlap	76	5.0^{***}	18^{NS}	$34/42^{NS}$
	5' Overlap		3.5^{NS}	18^{NS}	
Non-Overlapping	NA	322	2.2	57	NA

Figure S1. Relationship between gene overlap (arcsin-sqrt) and information length (ln). Gene overlap is the angle whose sine is the square root of the overlap as a proportion of the information length. Information length is the natural logarithm of genome length plus overlap length. Untransformed values for both axes are shown in the inset. Points are means for the following taxa.

1 - Acyrthosiphon pisum virus (n=1); 2 - Arenaviridae (n=11); 3 - Arteriviridae (n=4); 4 - Astroviridae (n=6); 5 - Barnaviridae (n=1); 6 - Beet western yellows ST9 associated virus (n=1); 7 - Benyvirus (n=2); 8 - Birnaviridae (n=5); 9 - Bornaviridae (n=1); 10 - Botrytis virus F (n=1); 11 - Botrytis virus X (n=1); 12 - Bromoviridae (n=22); 13 - Bunyaviridae (n=20); 14 - Caliciviridae (n=13); 15 - Caulimoviridae (n=23); 16 - Cheravirus (n=2); 17 - Chrysoviridae (n=1); 18 - Closteroviridae (n=16); 19 - Comoviridae (n=18); 20 - Coronaviridae (n=12); 21 - Cystoviridae (n=4); 22 - Diaporthe ambigua RNA virus 1 (n=1); 23 - Dicistroviridae (n=12); 24 - Endornavirus (n=1); 25 - Filoviridae (n=4); 26 - Flaviviridae (n=34); 27 - Flexiviridae (n=52); 28 - Furovirus (n=5); 29 - Fusarium graminearum dsRNA mycovirus-1 (n=1); 30 - Hepadnaviridae (n=10); 31 - Hepeviridae (n=1); 32 - Hordeivirus (n=1); 33 - Hypoviridae (n=4); 34 - Idaeovirus (n=1); 35 - Iflavirus (n=7); 36 - Leviviridae (n=8); 37 - Luteoviridae (n=17); 38 - Marnaviridae (n=1); 39 - Narnaviridae (n=8); 40 - Nodaviridae (n=8); 41 - Ophiovirus (n=3); 42 - Orthomyxoviridae (n=5); 43 - Oyster mushroom spherical virus (n=1); 44 - Paramyxoviridae (n=28); 45 - Partitiviridae (n=14); 46 - Pecluvirus (n=2); 47 - Picobirnavirus (n=1); 48 - Picornaviridae (n=31); 49 - Pomovirus (n=4); 50 - Potyviridae (n=54); 51 - Reoviridae (n=21); 52 - Retroviridae (n=40); 53 - Rhabdoviridae (n=17); 54 - Sclerophthora macrospora virus A (n=1); 55 - Sequiviridae (n=6); 56 - Sobemovirus (n=9); 57 - Tenuivirus (n=2); 58 - Tetraviridae (n=4); 59 - Thielaviopsis basicola dsRNA virus 1 (n=1); 60 - Tobamovirus (n=15); 61 - Tobravirus (n=3); 62 - Togaviridae (n=16); 63 - Tombusviridae (n=35); 64 - Totiviridae (n=20); 65 - Tymoviridae (n=12); 66 - Umbravirus (n=4).

Belshaw et al. Figure S1

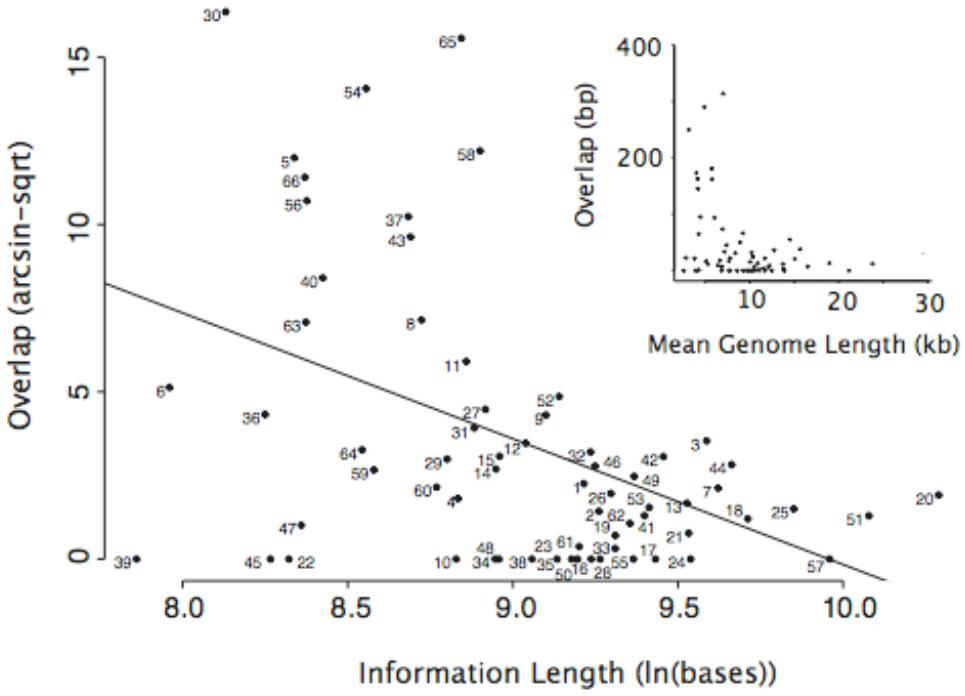
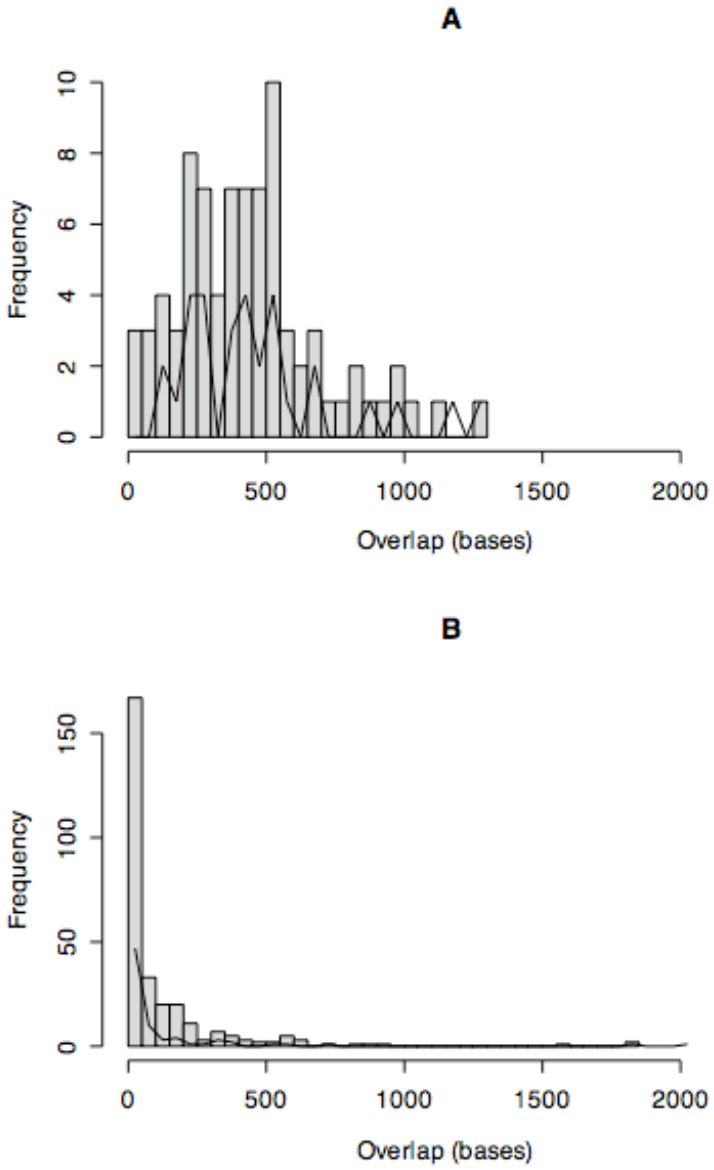


Figure S2. Observed overlap frequency distributions (each value is the mean of an homologous group): (A) Internal Overlaps. (B) Terminal Overlaps. Histograms represent all genes while lines represent a smaller dataset consisting only of genes with no potentially complicating factors such as splicing, internal frameshifting or multiple overlaps. Note the absence of very short Internal Overlaps in the smaller dataset (these are small overlapped components of larger spliced genes).



R script for fitness simulation (also given in Archive)

The following three scripts (which need to be cut and pasted) show simulations of the effect of increasing the proportion of the genome in an overlap on the fitness effect of mutations.

1. Basic script showing the effect of gene overlap on a multiplicative model of fitness.
2. Different models of the effect multiple mutations on fitness: comparing multiplicative to synergistic etc
3. Effect of reducing the per genome mutation rate on the effect of gene overlap with the basic multiplicative model

```
#---First script: for showing the effect of gene overlap on a multiplicative model of fitness --------

rm(list=ls())		# Delete all existing variables

m = 0.0001 # mutation rate per bp
g = 10000 # starting genome size (no overlap)
s = 0.1 # selection coefficient

fitness = NULL
prog_fit = NULL # fitness of progeny with increasing numbers of mutations
total_prog_fit = NULL #  fitness of progeny = sum of frequency of each mutant class of progeny multiplied by its fitness
prop = NULL

class_fitness <- function(n, p) { # function for determining total number of mutations: those that fall in regions of gene overlap count twice!
	prob_over = (p/2)/((1-p)+(p/2)) # probability of mutation falling in region of overlap
	for (k in 0:n) { # go through all possible combinations of 0 -> n mutations falling in regions of overlap
		#exp_freq = (exp(-prob_over)*(prob_over^k))/factorial(k)
		exp_freq = prob_over^k * (1-prob_over)^(n-k) * (factorial(n) / (factorial(k) * factorial(n - k))) # relative expected freqencies. factorial(n) / (factorial(k) * factorial(n - k)) is the binomial coefficient = number of combinations in which can achieve this number of muattions in overlap
		fitness[k+1] = exp_freq * exp( - s * ( (k*2) + (n-k) ) )
	}
	return(sum(fitness))
}

for(i in 0:100) {
	p = i/100 # just converting p from an integer to a proportion
	u = m * g * (1 - p/2) # adjust per genome mutation rate to allow for compression due to overlap
	freq = exp(-u) # need frequency of zero class at start
```

```
	prog_fit[1] = freq * 1 # = frequency of progeny with zero mutations multiplied by their fitness, which equals one (= exp(- s * 0 ))
	for(n in 1:10) { # no need to go beyond frequency of progeny with 5 mutations (proportion is < 0.001)
		freq = freq * (u/n)
		prog_fit[n+1] = freq * class_fitness(n, p) # multiply frequency of each class by its fitness

	}
	total_prog_fit[i+1] = sum(prog_fit)
	prop[i+1] = p # just for plotting
}

plot(prop, total_prog_fit, col=1, type="l",xlab="overlap", ylab="fitness", main="", frame.plot = FALSE)

#---Second script: for comparing different models for the impact of multiple mutations on fitness
rm(list=ls())		# Delete all existing variables

m = 0.0001 # mutation rate per bp
g = 10000 # starting genome size (no overlap)
s = 0.1 # selection coefficient
b = 0.01  # second selection coefficient for epistasis
fitness = NULL
prog_fit_mult = NULL # fitness of progeny with increasing numbers of mutations
total_prog_fit_mult = NULL #  fitness of progeny = sum of frequency of each mutant class of progeny multiplied by its fitness
prog_fit_mult2 = NULL # as above, but for alternative model of multiplicative fitness interactions
total_prog_fit_mult2 = NULL #  # as above, but for alternative model of multiplicative fitness interactions
prog_fit_ant_epi = NULL # as above, but for antagonostic epistasis
total_prog_fit_ant_epi = NULL # as above, but for antagonostic epistasis
prog_fit_syn_epi = NULL # as above, but for synergistic epistasis
total_prog_fit_syn_epi = NULL # as above, but for synergistic epistasis
prog_fit_add = NULL # as above, but for additive fitness
total_prog_fit_add = NULL # as above, but for additive fitness
prop = NULL

# basic multiplicative fitness
class_fitness_mult <- function(n, p) { # function for determining total number of mutations: those that fall in regions of gene overlap count twice!
	prob_over = (p/2)/((1-p)+(p/2)) # probability of mutation falling in region of overlap

	for (k in 0:n) { # go through all possible combinations of 0 -> n mutations falling in regions of overlap
```

```
		bin_coeff = factorial(n) / (factorial(k) * factorial(n - k)) # binomial coefficient =
number of combinations in which can achieve this number of muattions in overlap
		exp_freq = prob_over^k * (1-prob_over)^(n-k) * bin_coeff # relative expected
freqencies
		fitness[k+1] = exp_freq * exp( - s * ( (k*2) + (n-k) ) )
	}
	return(sum(fitness))
}

# alternative multiplicative fitness
class_fitness_mult2 <- function(n, p) { # function for determining total number of mutations:
those that fall in regions of gene overlap count twice!
	prob_over = (p/2)/((1-p)+(p/2)) # probability of mutation falling in region of overlap

	for (k in 0:n) { # go through all possible combinations of 0 -> n mutations falling in
regions of overlap
		bin_coeff = factorial(n) / (factorial(k) * factorial(n - k)) # binomial coefficient =
number of combinations in which can achieve this number of muattions in overlap
		exp_freq = prob_over^k * (1-prob_over)^(n-k) * bin_coeff # relative expected
freqencies
		fitness[k+1] = exp_freq * (1 - s)^((k*2) + (n-k))
	}
	return(sum(fitness))
}

# antagonistic epistasis
class_fitness_ant_epi <- function(n, p) {
	b = 0.01  # second selection coefficient for epistasis
	prob_over = (p/2)/((1-p)+(p/2))
	for (k in 0:n) { # go through all possible combinations of 0 -> n mutations falling in
regions of overlap
		bin_coeff = factorial(n) / (factorial(k) * factorial(n - k)) # binomial coefficient =
number of combinations in which can achieve this number of muattions in overlap
		exp_freq = prob_over^k * (1-prob_over)^(n-k) * bin_coeff # relative expected
freqencies
		fitness[k+1] = exp_freq *  exp(- s * ((k*2) + (n-k))  + b * ((k*2) + (n-k))^2)
	}
	return(sum(fitness))
}

# synergistic epistasis
class_fitness_syn_epi <- function(n, p) {
	b = 0.02  # second selection coefficient for epistasis
	prob_over = (p/2)/((1-p)+(p/2))
	for (k in 0:n) { # go through all possible combinations of 0 -> n mutations falling in
regions of overlap
```

```
		bin_coeff = factorial(n) / (factorial(k) * factorial(n - k)) # binomial coefficient = number of combinations in which can achieve this number of muattions in overlap
		exp_freq = prob_over^k * (1-prob_over)^(n-k) * bin_coeff # relative expected freqencies
		fitness[k+1] = exp_freq *   exp(- s * ((k*2) + (n-k))  - b * ((k*2) + (n-k))^2)
	}
	return(sum(fitness))
}

# additive epistasis
class_fitness_add <- function(n, p) {
	prob_over = (p/2)/((1-p)+(p/2))
	for (k in 0:n) { # go through all possible combinations of 0 -> n mutations falling in regions of overlap
		bin_coeff = factorial(n) / (factorial(k) * factorial(n - k)) # binomial coefficient = number of combinations in which can achieve this number of muattions in overlap
		exp_freq = prob_over^k * (1-prob_over)^(n-k) * bin_coeff # relative expected freqencies
		fitness[k+1] = exp_freq *   (1 - (s * ((k*2) + (n-k))))
	}
	return(sum(fitness))
}



for(i in 0:100) {
	p = i/100 # just converting p from an integer to a proportion
	u = m * g * (1 - p/2) # adjust per genome mutation rate due to overlap
	freq = exp(-u) # need frequency of zero class at start
	prog_fit_mult[1] = freq * 1 # = frequency of progeny with zero mutations multiplied by their fitness, which equals one (= exp(- s * 0 ))
	prog_fit_mult2[1] = freq * 1
	prog_fit_ant_epi[1] = freq * 1 # same for all fitness functions
	prog_fit_syn_epi[1] = freq * 1
	prog_fit_add[1] = freq * 1
	for(n in 1:10) { # no need to go beyond frequency of progeny with 5 mutations (proportion is < 0.001)
		freq = freq * (u/n)
		prog_fit_mult[n+1] = freq * class_fitness_mult(n, p) # multiply frequency of each class by its fitness
		prog_fit_mult2[n+1] = freq * class_fitness_mult2(n, p) # multiply frequency of each class by its fitness
		prog_fit_ant_epi[n+1] = freq * class_fitness_ant_epi(n, p) # multiply frequency of each class by its fitness
		prog_fit_syn_epi[n+1] = freq * class_fitness_syn_epi(n, p) # multiply frequency of each class by its fitness
```

```
                prog_fit_add[n+1] = freq * class_fitness_add(n, p) # multiply frequency of each
class by its fitness


        }
        total_prog_fit_mult[i+1] = sum(prog_fit_mult)
        total_prog_fit_mult2[i+1] = sum(prog_fit_mult2)
        total_prog_fit_ant_epi[i+1] = sum(prog_fit_ant_epi)
        total_prog_fit_syn_epi[i+1] = sum(prog_fit_syn_epi)
        total_prog_fit_add[i+1] = sum(prog_fit_add)
        prop[i+1] = p # just for plotting

}

plot(prop, total_prog_fit_mult/total_prog_fit_mult[1], col=1, type="l", ylim = range(0.98, 1.02),
xlab="Prop overlap", ylab="Fitness", main="", frame.plot = FALSE)
points(prop, total_prog_fit_mult2/total_prog_fit_mult2[1], col=2, type="l")
points(prop, total_prog_fit_ant_epi/total_prog_fit_ant_epi[1], col=3, type="l")
points(prop, total_prog_fit_syn_epi/total_prog_fit_syn_epi[1], col=4, type="l")
points(prop, total_prog_fit_add/total_prog_fit_add[1], col=5, type="l")
legend(x="topright", legend=c("mult", "mult2", "ant_epi", "syn_epi", "add"), fill= 1:5, bty="n")

#---Third script: for showing the effect of reducing per genome mutation rate on the basic
multiplicative model

rm(list=ls())               # Delete all existing variables

s = 0.1 # selection coefficient
fitness = NULL
k.values = NULL
prog_fit = NULL # fitness of progeny with increasing numbers of mutations
total_prog_fit = NULL #  fitness of progeny = sum of frequency of each mutant class of progeny
multiplied by its fitness
prop = NULL
slope = NULL
mut_rate = NULL
start_m = 0.0001 # starting mutation rate per bp
g = 10000 # genome size (no overlap)

class_fitness <- function(n, p) { # function for determining total number of mutations: those that
fall in regions of gene overlap count twice!
        prob_over = (p/2)/((1-p)+(p/2)) # probability of mutation falling in region of overlap
        for (k in 0:n) { # go through all possible combinations of 0 -> n mutations falling in
regions of overlap
                bin_coeff = factorial(n) / (factorial(k) * factorial(n - k)) # binomial coefficient =
number of combinations in which can achieve this number of muattions in overlap
```

```
                exp_freq = prob_over^k * (1-prob_over)^(n-k) * bin_coeff # relative expected
freqencies
                fitness[k+1] = exp_freq * exp( - s * ( (k*2) + (n-k) ) )
        }
        return(sum(fitness))
}

for(j in 1:100) {
        m = start_m/j # reducing mutation rate
        for(i in 0:100) {
                p = i/100 # just converting p from an integer to a proportion
                u = m * g * (1 - p/2)  # adjust per genome mutation rate due to overlap
                freq = exp(-u) # need frequency of zero class at start
                prog_fit[1] = freq * 1 # = frequency of progeny with zero mutations multiplied by
their fitness, which equals one (= exp(- s * 0 ))
                for(n in 1:5) { # no need to go beyond frequency of progeny with 5 mutations
(proportion is < 0.001)
                        freq = freq * (u/n)
                        prog_fit[n+1] = freq * class_fitness(n, p) # multiply frequency of each
class by its fitness
                }
                total_prog_fit[i+1] = sum(prog_fit)
                prop[i+1] = p
        }
        saved.lm = lm(total_prog_fit ~ prop)
        slope[j] = saved.lm$coefficients[2]
        mut_rate[j] = m
}


plot(mut_rate, slope, col=1,  type="l", xlab="mut rate", ylab="slope", main="", frame.plot =
FALSE)
```